RELIABLE ETHERNET

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Aleksandr Movsesyan

June 2011

COMMITTEE MEMBERSHIP

TITLE:	Reliable Ethernet
AUTHOR:	Aleksandr Movsesyan
DATE SUBMITTED:	June 2011
COMMITTEE CHAIR:	Hugh M. Smith, Ph.D.
COMMITTEE MEMBER:	John Bellardo, Ph.D.
COMMITTEE MEMBER:	Phillip L. Nico, Ph.D.

ABSTRACT

Reliable Ethernet

Aleksandr Movsesyan

Networks within data centers, such as connections between servers and disk arrays, need lossless flow control allowing all packets to move quickly through the network to reach their destination. This paper proposes a new algorithm for congestion control to satisfy the needs of such networks and to answer the question: Is it possible to provide circuit-less reliability and flow control in an Ethernet network? TCP uses an end-to-end congestion control algorithm, which is based on end-to-end round trip time (RTT). Therefore its flow control and error detection/correction approach is dependent on end-to-end RTT. Other approaches utilize specialized data link layer networks such as InfiniBand and Fibre Channel to provide network reliability.

The algorithm proposed in this thesis builds on the ubiquitous Ethernet protocol to provide reliability at the data link layer without the overhead and cost of the specialized networks or the delay induced by TCP's end-to-end approach. This approach requires modifications to the Ethernet switches to implement a back pressure based flow control algorithm. This back pressure algorithm utilizes a modified version of the Random Early Detection (RED) algorithm to detect congestion.

Our simulation results show that the algorithm can quickly recover from congestion and that the average latency of the network is close to the average latency when no congestion is present. With correct threshold and alpha values, buffer sizes in the network and on the source nodes can be kept small to allow little needed additional hardware to implement the system.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Introduction

This research presents an algorithm to provide reliable end-to-end communication over Ethernet without the overhead and cost of specialized networks or the delay induced by TCP. Currently in systems requiring reliability and low-latency, such as servers connected to disk arrays, specialized networks such as InfiniBand and Fibre Channel are required.

Our research presents an algorithm which builds off of the ubiquitous Ethernet protocol, but provides reliability to the data link layer. This approach eliminates the inefficiencies of the end-to-end flow control and error recovery specialized protocols like InfiniBand and Fibre Channel.

The solution proposed in this thesis uses back pressure and Random Early Detection (RED) to provide for flow control and error recovery. In addition Ethernet processing is modified to respond to congestion using an Additive-Increase Multiplicative-Decrease (AIMD) algorithm.

In our solution, each node in the network keeps two buffers for each of the outgoing links, and ACK buffer and a send buffer. The ACK buffer is used to store packets that have been sent and are awaiting acknowledgements from the next downstream device. A packet stays in the ACK buffer until it receives an acknowledgement from the upstream node at which point the packet is removed. The send buffer is used to buffer packets prior to transmitting them to the next

hop. The switches calculate average buffer size used by the RED algorithm based on the utilization of these buffers.

Switches inform source nodes of congestion that is occurring based on the RED algorithm. Upon receiving a congestion packet, a source node will decrease its sending rate using an Additive Increase Multiplicative Decrease (AIMD) approach. This reduction at the source node only happens for traffic destined to a particular destination on the path that the congestion occurred. This allows the source node to continue sending traffic to other destination nodes while decreasing the rate on the congested port.

When a node, either a source node or switch is ready to transmit a packet it first checks the ACK buffer. If the packet at the head of the ACK buffer is older than the expected round trip time (RTT) to the next node, this packet will be retransmitted. Otherwise the packet at the head of the send buffer will be transmitted. Once a packet is transmitted from either buffer it is placed at the tail of the ACK buffer.

Initial tests of the algorithm using network simulation software, show that the algorithm recovers from congestion quickly and only punishes specific source nodes sending to specific destinations based on link congestion. By doing this, the source nodes are able to continue to send traffic to other destinations while

only backing off on congested routes. The simulations also show that with the correct threshold values, buffer sizes on the switch of each outgoing link can be kept small, even when congestion occurs. The solution is thus a viable one for the given problem with a few minor tweaks for testing on physical devices.

As described above this approach takes advantage of the low latency between switches. This is in comparison to TCP's dependence on end-to-end latency. This lower latency allows our approach to keep buffers small in the switches. In addition, since the approach does not require utilizing specialized networks, such as InfiniBand and Fibre Channel, which require upfront configuration and the maintenance of virtual circuits.

The rest of this paper is structured as follows. Section 2 describes the various algorithms that are alternatives to the one given in this paper and the current background research that has been done to solve this problem. Section 3 describes the algorithm proposed in detail and Section 4 shows the simulation and results of the created algorithm. Section 5 describes future work that must be done to fully flesh out this solution. Finally, Section 6 gives the conclusion.

2 Background

2.1 TCP

One main transfer protocol is the Transport Control Protocol (TCP) which is the primary transport protocol for the Internet suite. “TCP provides reliable, in-sequence delivery of a full-duplex stream of octets (8-bit bytes). TCP is used by those applications needing reliable, connection-oriented transport service” [1].

Combining the TCP algorithm with other protocols such as RED, ECN, or SACK allows the protocol to perform better in any network. Due to this, significant research has been done to improve algorithms such as these to have on top of TCP.

2.1.1 RED

Random early detection (RED) is an algorithm that actively manages the buffering in a device by using average queue length [2]. The RED algorithm calculates the average buffer size in a device and then drops or marks a packet with some non-zero probability, if the average is between the minimum and maximum thresholds and 100% probability if the average is above the maximum threshold.

Improving the RED algorithm allows networks to detect and avoid congestion quicker than standard RED currently does. Standard RED is very

sensitive to a network environment due to the fixed threshold settings [8]. These parameters are configured specifically for each network. Without setting these values correctly, RED's performance would be hindered in that network. To improve this, RED research has looked at dynamically adapting the parameters to network changes.

Multiple improvements to RED have been proposed to solve this problem. Solutions, such as Adaptive RED (ARED) found in [9] and buffer occupation with RED (BORED) found in [10], have been shown to increase the performance versus standard RED by adjusting the drop probability and the thresholds respectively. With a dynamic buffer management method, a router achieves higher link utilization, higher throughput, and a lower drop rate [11].

Another proposed solution is to combine the ARED and BORED in order to get the benefits of both. This new algorithm is called BO-ARED. It aims to solve the sensitivity problem of the RED parameters to network changes with minimal changes to this algorithm [10].

Using the BO-ARED algorithm, the RED parameters should be set when loading the algorithm on a router so the algorithm knows how much space it is able to work with. BO-ARED has the buffer occupation as the rate that all of the current packets occupy the buffer [10]. "$B_r = Q_c / B_f$ where, Q_c is the instantaneous queue size; B_f is the setting of the buffer size. And B_r is the current

buffer occupation in this paper" [10]. w_q, the queue weight of the instantaneous queue size, should also be dynamic in order to cope with the changing network. "When w_q is too low, then the estimated average queue size probably responds too slowly to transient congestion. When w_q grows too high, the estimated average queue size tracks the instantaneous queue size too closely" [10].

In these terms the BO-ARED algorithm adds minor tweaks to the current RED algorithm in order to adapt the min (min_{th}) and max (max_{th}) thresholds, w_q, and other variables for RED. This allows RED to constantly adapt to the network and avoid static reconfiguration once things change. The initial configuration would be the only needed configuration in order to get the benefit of the newly proposed algorithm.

"The analysis and simulations all demonstrate that the BO-ARED algorithm can be suitable to network variation rapidly and reduces both drop rate and delay time" [10]. There are three ways that this algorithm improves RED. First, as described above, the algorithm doesn't need the parameters to be set specifically to the network [10]. Second, due to smoother average queue plots, BO-ARED could get both lower drop rate and low average delay time in busty-traffic [10]. Finally, "BO-ARED improve the weakness of BORED that it reduces the long delay time and stable the queue under the heavy loads; otherwise, BORED decreases the high drop rate of ARED" [10].

The flow control solution we propose in this thesis utilizes standard RED for active queue management. Our implementation suffers from the normal parameter sensitivity of the standard RED approach. The implantation suggested by BORED, ARED, and BO-ARED would also benefit our approach and are discussed in our future work section.

2.1.2 ECN

Explicit Congestion Notification (ECN) is a mechanism to provide fast feed-back from the routers to the ECN-enabled TCP source about impending congestion. "The three entities in the network, viz. the source, the router, and the receiver need to cooperate in the successful application of ECN to control congestion" [3]. The source needs to inform the routers that it is capable of responding to ECN notification, the routers need to be able to deliver the congestion notification through the receiver to the source, and the receiver needs to echo the ECN information to the sender.

ECN uses two bits in the IP header and two bits in the TCP header to identify potential congestions. The router uses RED to identify potential congestion, but instead of dropping packets, it sets one of the ECN bits in the IP header. "The receiver, upon receiving a marked IP packet, echoes this information back to the sender by setting a bit in the TCP header of the acknowledgement" [3]. Once the sender receives a packet with a congestion

notification, the sender will adjust its sending rate. This allows for the congestion notification to be delivered without the need to drop the packet and the consequent loss of throughput.

Research into explicit congestion control (ECN) has also been done in order to improve congestion control within networks. ECN is another algorithm that can be mixed in with other RED and Back Pressure algorithms to improve the performance of our algorithm.

A new algorithm is proposed in [12] that has a new Active Queue Management (AQM) called NEWQUE that supports ECN. The objective of this new algorithm is to improve performance of congested routers by keeping link utilization high and stable, queue sizes stable, and packet drop rate low. Simulation of the algorithm shows that it outperforms the peer AQM schemes in terms of packet loss, link utilization, and buffer fluctuation [12].

"NEWQUE uses the flow arrival rate, the link capacity and link utilization history to manage congestion rather than on the instantaneous or average queue lengths" [12]. Within this algorithm only a single marking probability is maintained. This probability is incremented when the flow arrival rate is greater than or equal to the link capacity and decremented when the link is idle or the flow arrival rate is less than the link capacity [12]. This allows for the algorithm to determine the correct rate to mark packets for congestion notification.

NEWQUE is another algorithm that can be used in conjunction with the proposed solution in this paper. The proposed algorithm in this paper could use the best mix to find congestion and use the algorithm described in this paper in order to notify of this congestion.

2.1.3 SACK

Selective Acknowledgment (SACK) is a mechanism where the receiver informs the sender about all segments that have arrived successfully, so the sender only needs to retransmit segments which have actually been lost [4]. This solves the problem with TCP and poor performance when multiple packets are lost from one window of data, since TCP can only inform the sender of a single lost packet per round trip time. TCP SACK is fully defined in [4]. Using SACK, TCP can handle multiple segment losses within a single window without incurring a retransmission timeout. "It also provides additional information about congestion state, helping TCP recover faster" [5].

There has been much research done into improving the SACK option for TCP. Most of this research is looking at wireless networks. Research has been done to improve the SACK option on top of TCP to detect a lost retransmission in order for it not to happen again. This research is found in [13].

2.2 Back Pressure

Many algorithms use back pressure within their implementations to allow congestion to reach the source node. The back pressure algorithm is a way of notifying the sender that congestion has occurred. When a link in the system gets congested, the buffer that node is holding for that link begins to fill up. Once the buffer fills up, all nodes sending to that link will then have their buffers begin to fill up. Once a buffer starts to fill, the node needs to then tell the previous node to back off. These two steps will reoccur until the sending source node is notified that there is congestion in the system.

This algorithm avoids notifying source nodes to back off if minor congestion occurs and is cleared up quickly. In that case, only a few buffers will fill up. Once the network becomes uncongested, the nodes will send traffic as usual clearing out the buffers.

2.3 XCP

Another solution looking at a global problem of TCP to address congestion control in high latency networks is eXplicit Control Protocol (XCP). XCP is a generalization of the Explicit Congestion Notification proposal (ECN) [16].

In XCP, there are a few distinct parts that make the system work. Each sender will need to keep track of their own window size and round trip time in

order to be able to inform the network of its state. This system works a lot like TCP since it is a window based congestion control system.

The first component of XCP is the congestion header. The congestion header is a newly proposed header that will be used to inform the system of the current status of the sender. The header will consist of three pieces of information: the current window size of the sender (cwnd), the current round trip time of the sender (rtt), and feedback from the system. The header can be seen in Figure 1.

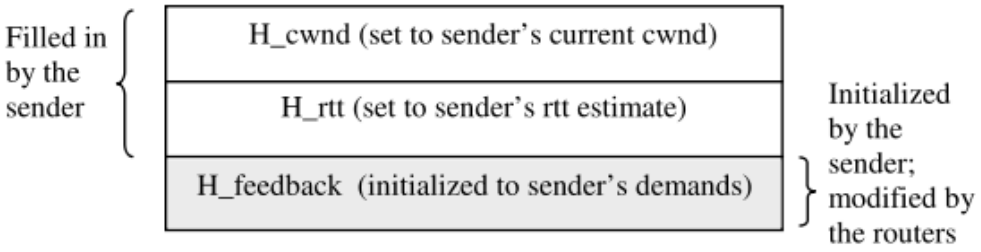

Figure 1: Congestion header in the XCP system [16]

"As with TCP, an XCP sender maintains a congestion window of the outstanding packets, cwnd, and an estimate of the round trip time rtt" [16]. Before a packet is routed through the system, the sender attaches a congestion header to the packet and sets the H_cwnd field to its current cwnd and H_rtt to its current rtt. In the header of the first packet in the flow, H_rtt is set to zero to indicate to the system that the rtt is not yet known. The sender then initializes H_feedback to request its desired window increase. The desired window increase is calculated by taking the desired rate r, multiplying it by the rtt, subtracting

cwnd from the product, and then dividing by the number of packets in the current congestion window ((r * rtt - cwnd)/# packets).

"If bandwidth is available, this initialization allows the sender to reach the desired rate after one rtt" [16]. As the packet is routed through the system this field is adjusted by the various routers. After receiving the feedback, the sender changes its window size based on the feedback given and continues to send traffic.

The third part of the XCP system is a receiver. This receiver works identically to a TCP receiver, except that when acknowledging a packet, it copies the congestion header from the data packet to the acknowledgement packet.

Fourth, the system needs a special router that has the controls inside. The main job of the router is to compute the feedback to cause the system to converge to optimal efficiency and min-max fairness [16]. This creates an environment where no packets will be dropped. XCP works on top of other solutions with dropping policies such as DropTail, RED, or AVQ, but does not allow the buffer to get large enough where packets need to be dropped at all.

To compute the feedback, an XCP router has two controllers: an efficiency controller and a fairness controller. "Both of these compute estimates over the average RTT of the flows traversing the link, which will smooth the burstiness of

a window-based control protocol" [16]. This allows for estimates to not be sluggish or erroneous due to too slow or too fast responses respectively. These controllers make a single control decision every RTT to be able to observe the results of the previous decision.

The efficiency controller (EC) has the purpose of maximizing link utilization while minimizing drop rate and persistent queues. It only looks at aggregate traffic and leaves the fairness issues to the fairness controller. The EC computes a desired increase or decrease in the number of bytes that aggregate traffic transmits in a control interval, RTT.

The job of the fairness controller (FC) is to apportion the feedback to individual packets to achieve fairness. The FC relies on Additive-Increase Multiplicative-Decrease (AIMD), just as TCP, to converge to fairness. The FC figures out the per packet change and adjusts the feedback field in the XCP header that will eventually reach the sending node, who will make adjustments.

XCP takes a very aggressive approach of backing off in order to keep buffer sizes constant and the number of dropped packets low. By doing this, XCP is able to see a packet drop rate of about one per thousand packets and no decrease in performance over TCP. Instead, there are increases in performance over very high latency networks.

This approach is similar to the one proposed in this paper with a few key differences. First, the proposed system in this paper is its own congestion and error correction algorithm that is stand-alone. XCP is built on top of other systems that are already working. Secondly, XCP requires new routers that will be able to read and interpret the newly added headers and change them as needed. Our approach uses switches to be able to send acknowledgements back to the previous node or the sender.

Another key difference is the time that it takes for the sender to be notified to adjust for congestion. XCP works similarly to TCP where it sends the packet across the network and then informs the sender what to do from the acknowledgement packet sent from the receiving node. Our solution, on the other hand, informs the sender directly from the node where the congestion occurred. This allows for the worst case scenario to be one hop less than XCP and the best case scenario to be much better since congestion could occur after the first hop.

Finally, the key difference between the systems is what we are trying to fix. XCP is looking to solve the problems that will occur with TCP as we begin to move toward longer latency networks such as satellite and mobile. We are looking at very low latency networks where optimizations can be made in order to take advantage of characteristics in the system.

2.4 InfiniBand

InfiniBand uses switched fabric architecture in order to break the bandwidth and fan-out limitations of the PCI bus. Nodes in this architecture are connected using the InfiniBand fabric. The nodes represent either a host device such as a server or an I/O device such as RAID subsystem. The fabric itself can be constructed from any number of InfiniBand switches or routers.

"Each of the connections between nodes, switches, and routers is a point-to-point, serial connection" [6]. This difference allows for a number of benefits.

- Due to being a serial connection, it only requires four connections of the PCI bus.
- Due to the point-to-point nature of the protocol, the connection provides full capacity of the connection to the two endpoints; the link is dedicated to these endpoints.
- Due to the short length of the connection, higher bandwidth can be achieved.

The InfiniBand defines raw bandwidth where 1x connection provides a 2.5Gb per second line. [6] There is also 4x and 12x versions which multiply the connection speed by the given amount. This allows the architecture to achieve much higher data transfer rates than is physically possible with the shared bus architecture.

There has been a lot of research placed into making InfiniBand a better solution for the Internet domain. InfiniBand "provides high bandwidth, expandability, and scalability" [17]. The research done in [17] is geared toward making a cheaper and more effective link layer for the InfiniBand architecture.

The paper presents a design and implementation of the link layer of an InfiniBand HCA (Host Channel Adapter). This implementation uses six virtual lanes (VLs), three to send and three to receive, to communicate between the link and transport layers. To be a better solution the receiver uses a high speed packet buffering architecture with a FIFO (first in first out) circuit.

This architecture "enables the efficient utilization of the InfiniBand bandwidth and the reduction of hardware costs as well as power consumption" [17]. This newly proposed architecture can also be applied to state-of-the-art I/O standards that have high-speed switched fabric architectures such as Gigabit Ethernet and Fibre Channel.

The research is different from the one proposed in our paper in both the approach taken and the benefits provided by their approach. The more efficient link layer is a great architecture to be applied to various I/O standards, but it still creates a more expensive solution than Ethernet can offer. Also, this new architecture needs to be configured differently for each of the different standards

and the chip that they have created would need to be changed for each of these architectures as well.

The main focus of the research is InfiniBand; and even with the cost reductions that the authors claim, Ethernet would still be cheaper. Our proposed solution only needs to add a little hardware to the current Ethernet architecture to allow it to work properly, which allows for easier adoption.

Other research on the InfiniBand architecture is working toward creating an algorithm that will manage congestion within the system. InfiniBand cannot drop packets to deal with congestion and so switch buffers can fill up, block upstream switches and even choke flows that are not contending for the congested link [18]. This can cause the undesired effect of congestion spreading where other sources are affected by congestion even though they are not causing it. "Congestion spreading, also known as tree saturation occurs when a switch buffer fills up and blocks the buffers upstream" [18].

To alleviate the problem of congestion spreading, a TCP-like congestion control where the switches track flow and mark source packets that are causing congestion in the packet header to inform the destination device. Once a packet is marked for congestion, the destination device notifies the source device of the congestion. The source then adjusts its packet injection based on "a novel hybrid

mechanism that combines the advantages of rate and window control. Adjustment policies are designed to enable high throughput while preventing starvation" [18].

The marking of congestion happens once congestion occurs and the buffer on a specific link is full. The congestion bit in the packet header of all packets in the buffer gets set to 1 so their sources get notified to adjust their packet injection. Adjustment of the packet injection happens for a specific flow. Once a source is informed of congestion it adjusts the rate limit and window size.

Noncompliant switches will not be able to mark congestion packets, and so congestion spreading may occur on these nodes, but all traffic will still work on noncompliant switches. Due to this problem, switching to the system will need to happen all at once if congestion spreading is to be avoided. Governments and data centers cannot have congestion spreading occur and thus would need to switch their entire system to use this algorithm.

This research was done to introduce a congestion algorithm into the InfiniBand architecture before there was one present. It has a similar algorithm to the one proposed in this paper, but with a few key differences. First, the destination must notify the sender of congestion in this algorithm while in the proposed algorithm the switch directly notifies the sender. Our approach allows for a faster response from the sender and allows for packets not to be dropped within an Ethernet setting.

Second, noncompliant switches and end nodes cause problems for the implementation while noncompliant end nodes would cause problems for the proposed solution. In the algorithm proposed in this paper, if there are noncompliant switches, then the network could not guarantee error free packets, but would still be able to handle congestion.

Both solutions strive to handle congestion in a timely manner with the least amount of flows affected. In other words, both only reduce the rate on sources that are causing the problem and do not punish everyone as in the current TCP solution.

2.5 Fibre Channel

Another protocol used for fast error recovery and congestion detection is Fibre Channel. "Fibre Channel is a highly reliable, gigabit, serial interconnect technology" [7]. This technology allows concurrent communications among various storage devices using protocols such as Small Computer System Interface (SCSI) and Internet Protocol (IP). Fibre Channel operates at data rates from 1 Gbps up to 10 Gbps [7]. Due to its high reliability and speed, this technology is used by the commercial industry for Storage Area Network (SAN) applications.

Fibre Channel networks consist of nodes connected by an interconnection scheme called a topology. Three different topologies are supported: point-to-

point, arbitrated loop, and switched fabric [7]. This technology communicates by sending frames through the system. "A frame is made up of transmission words, which contain 4 bytes each. A frame contains a start-of-frame delimiter (4 bytes), a header of 6 transmission words, an optional payload up to 528 transmission words, a 4-byte long Cyclic Redundancy Check (CRC), and an end-of-frame delimiter" [7].

The Fibre Channel standard has multiple ways to deliver frames referred to as Classes of Service to support the needs of a wide variety of applications and data types. This allows the technology to be more flexible in the type of network it can support.

Another competitor in the realm of data center networks is Fibre Channel. Research has also been done to improve this technology. "Fibre Channel was originally conceived as a single, high-bandwidth, multi-protocol data transport that could meet a variety of data communications needs. Over time, it has been widely regarded as the optimal data communications solution for storage environments due to its high-bandwidth, flexibility and reliability" [19].

Fibre Channel Data Communication Service (FCDCS) is proposed in [19] and operates with QLA2342 (2Gb/sec) Fibre Chanel Host Bus Adapters. FCDCS "is an optimized lightweight protocol that implements a simple peer-to-peer data transfers as a custom FC-4 specifications for the Upper Layer Protocols (ULPs)"

[19]. The algorithm minimizes the CPU load by bypassing host memory and directly transferring data into and out of user buffers. Applications transmit and receive data over Fibre Channel that connects the two nodes. "FCDCS also eliminates the relatively small data packets in IP, enabling large data transfers in a single Fibre Channel exchange and further increasing data throughput" [19].

This algorithm serves a different purpose than the one provided in this paper. It enhances Fibre Channel and doesn't look into congestion control. This would allow Fibre Channel to become a better solution where the system is currently used.

Most research in Fibre Channel has not been toward improving the congestion control. Fibre Channel has an effective system that allows fast data transfer that is beneficial to the customer at hand [19]. The algorithm proposed in this paper would provide similar speed benefits for this customer.

2.6 UETS/EFR

Due to the high importance of reliable Ethernet, the problem of data centers needing lossless flow control has been approached by many with varying degrees of success. Recently, a new architecture has been proposed to solve this problem called Universal Ethernet Telecommunications Service (UETS). UETS is "a highly scalable new dual stack architecture, networking protocol, and addressing schema, which allows the creating and delivery of new services with

greatly improved security and robustness to fully exploit the optional infrastructure of the next generation network" [14].

The proposed UTES approach allows for a new technique of routing called Ethernet Fabric Routing (EFR). EFR redefines the way packets are sent. In normal systems, a layer three based packet routing system is used. EFR changes this into a fully layer two approach where all packets are routed using MAC addresses. It uses local MAC addresses with a universal/local bit set in its packets to route the packet to the correct MAC address. This allows duplicate MAC addresses to be used in the network and packets to still be routed to the correct locations.

Switching can then be done by using the Banyan Matrix or similar switching fabrics. Using such a system would allow for the removal of routers and all other devices. The only hardware needed for the system is switches and end nodes that know of the system and can react to all packets that are sent across the wire as defined in [14].

The UETS architecture is composed of UETS network nodes (Central Universal Ethernet), network terminals (NTE/TRUE), and end nodes (TUE). The end nodes can have a mixture of TCP and Logical Link Control (LLC). LLC is the main way this new system routes packets, since it knows of the entire network and links the various switches.

This system allows for the use of regular packets at end nodes, since the end nodes can translate the packets into the desired ones used by the system. This will allow the architecture to be adapted faster since not all networks must switch immediately; rather only small segments at a time need to be switched.

The Central Universal Ethernet (CUE) is a new network node concept created for the UETS architecture. These node use hierarchical local (U/L bit = 1) addresses linked to physical port IDs. The frame switching and routing is based on local MAC destination addresses. This node routes packets in a way close to what BGP currently does. The CUE node is assigned an address prefix and a bit mask of variable length. All packets with the assigned address prefix will be routed to that node. Once the node receives a packet with the prefix, it applies the bit mask to the destination address and selects the port according to the value of the bit group as shown in Figure 2.

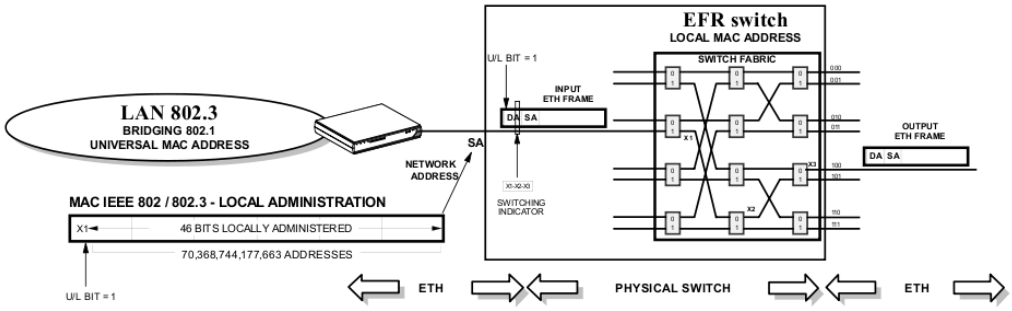

Figure 2: CUE packet routing [15]

Next, the NTE is responsible for the conversion between the current Internet architecture and UETS. NTE performs address translation (ENAT) or encapsulation between 802.1 universal (UMAC) and UETS local (LMAC) addresses. It also intercepts DNS requests and conveys them to the EDNS service as shown in Figure 3.

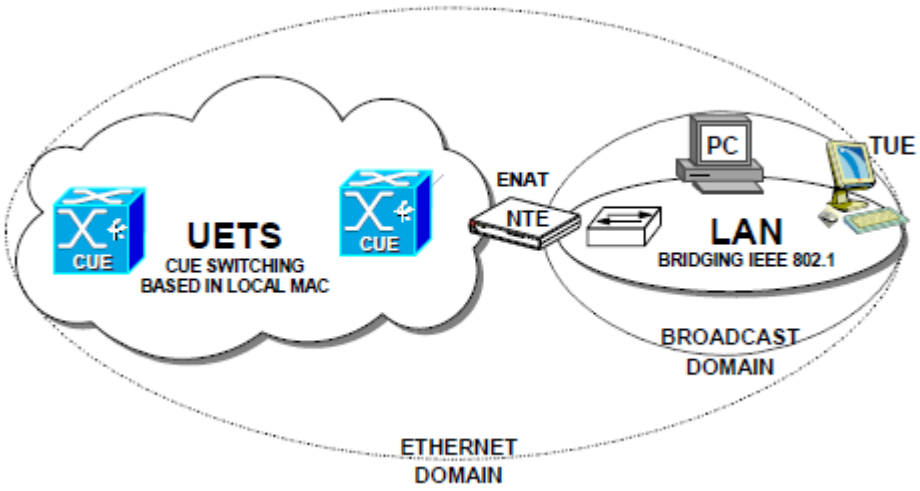

Figure 3: An example of a NTE node [15]

This new architecture allows for a solution to the reliability problems within Ethernet that is secure, is scalable, has a low cost/performance ratio, and is compatible with existing Ethernet and IP networks. As described above, this system can be placed in modularly while allowing the current Ethernet and IP networks to continue working. At each intersection between an Ethernet or IP network and a UETS network, a NTE node would need to be placed that would allow for the conversion of packets between the two different networks.

The solution is scalable to the entire world since you can reuse certain MAC addresses and thus need fewer addresses to handle all users in the world. This is very helpful as we are soon to run out of addresses in the IPv4 space and need to either switch to IPv6 or to a new networking solution such as UETS/EFR.

The best part of the solution is that it is secure. The responsibility for assigning MAC addresses is placed on the owners of the switches, which would most likely be Internet Service Providers (ISPs), and be hidden from the users. This takes away many attacks that are possible in current networks such as man in the middle attacks and address spoofing.

Also, because the solution removes TCP and IP protocols completely within the network, it allows for the solution to be much faster. Current networks are limited to the speeds of TCP and IP, while this solution would be only limited to hardware switching speeds, which are at least an order of magnitude faster.

The UETS solution is trying to solve a few worldwide problems and is trying to be the next network solution. This is a bit different than our proposed algorithm, since we are looking to improve performance in a low latency environment.

The UETS solution requires a total restructuring of the Internet which is currently not set up to allow for the new types of nodes that are present within the architecture. Our solution only requires minor modifications to switch devices.

Another aspect differentiating our solution and the UETS solution is our definition of reliable. UETS is working on a reliable Ethernet solution that is geared toward security which they believe to be the most important aspect of reliability. In contrast, we are more focused on delivering error free packets within a network in a timely manner.

3 Algorithm

To solve the problem of reliable Ethernet in terms of flow control and error detection within a system, an algorithm was created by combining a few existing techniques. The algorithm that was created can notify senders of congestion to ensure no packets are dropped and can guarantee that a packet will be transmitted from sender to receiver error free. The algorithm modifies the switches to utilize features of a Hop-by-Hop Back Pressure algorithm and Random Early Detection (RED).

3.1 Description of the Algorithm

In this algorithm, source nodes, e.g. user's computers, create packets and send them off to a destination node. Source nodes are modified to reduce their transmission to a specific destination when they receive a congestion packet from a switch. After receiving a congestion packet**,** the source node backs off and sends at a lowered rate to the specified destination node. Then in order to probe the network for available bandwidth, the source node increases its sending rate linearly. This is similar to the additive increase/multiplicative-decrease (AIMD) algorithm used by TCP.

Every node in the system keeps two buffers. The first buffer is called the sending buffer which is where packets are placed to be put on the wire. The second buffer is the ACK buffer which keeps track of packets that have already been sent and are awaiting ACK packets from the next upstream switch. When a

switch is ready to send a packet on an interface, it first checks the ACK buffer. If the packet at the front of this buffer is sufficiently old (e.g. two times the round trip time) this packet is sent, otherwise the packet at the front of the sending buffer is transmitted.

Once a packet is injected into the network, each switch, along the path to the destination node, figures if the packet can be accepted by that switch. There are four outcomes that can occur when a packet gets to a switch. First, the CRC can be incorrect in which case something occurred with the packet during transmission. Second, the packets can arrive out of order and hence a packet was lost in transmission. Third, either due to the buffer being full or the RED threshold, the algorithm can reject the packet by just dropping it within the switch. And lastly, the packet can be received by the switch without any problems or errors.

If a packet arrives out of order, a NAK will be created for the packet that was expected by the node. This NAK will be transmitted to the upstream node. When an upstream node receives a NAK packet, it retrieves the NAKed packet from its ACK buffer and retransmits the packet.

When receiving a packet, the switch will see if there is room for the packet in the sending buffer by applying the RED algorithm and ensuring that the average buffer size is below a set threshold value. The sending and the ACK

buffers are used in the calculation of the average. This is done by calculating the RED average for each of the buffers separately and then taking the maximum. If the RED average is less than the minimum threshold, the packet is placed in the sending buffer and an ACK packet is sent to the previous node. Otherwise, if the average buffer size is greater than the minimum threshold, the switch creates a congestion packet that is sent to the source of the packet.

It is possible that an ACK or NAK packet sent from a downstream node to an upstream node is lost. To recover from this situation, prior to sending a packet, the node first checks the packets in the ACK buffer. If the packet at the top of the ACK buffer is older than two times the round-trip time to the downstream node, the switch will retransmit this packet from the ACK buffer.

Conditions	**Receiving switch response**	**Upstream switch response**
Packet arrives out of order	NACK	Resend packet
Packet arrives and the average sending/ACK buffer is less than the minimum threshold and is accepted	ACK	Delete packet from ACK Queue
The average sending/ACK buffer size is greater than the maximum threshold and packet is rejected	Congestion Packet	Packet resent after 2*RTT
The average sending/ACK buffer size is greater than the minimum threshold with a non-zero probability and is rejected	Congestion Packet	Packet resent after 2*RTT

Table 1: Possible outcomes on receiving switch

This algorithm uses RED to determine when to reject a packet. The RED average queue length is recalculated every time a packet is received by using the formula of avg(n) = (alpha * numberOfPacketsInBuffer)+ ((1 - alpha) * avg(n-1))

[20]. Average queue length is calculated on both the ACK and sending buffers and the maximum of the two numbers is used for the current average of the outgoing link. Whenever the average of the buffer is greater than the RED minimum threshold and less than the maximum threshold for the RED algorithm, there is a non-zero chance that the packet will be dropped. This is determined by

((MAXTHRESHOLD - MINTHRESHOLD) * 1.0) /

(PMAX * (buffer average - MINTHRESHOLD))

and then taking one over this number minus the count since last drop [20]. The values for MAXTHRESHOLD, MINTHRESHOLD, and PMAX are described later in the results section as they pertain to the algorithm. If the average queue length is greater than the maximum threshold, the packet is always dropped.

When a switch receives a congestion packet, it forwards this congestion packet upstream to the source node. When the source node receives a congestion packet, it reduces its sending rate to the destination indicated in the congestion packet. In order to allow the source node to recover from this rate reduction to a particular destination, the source node increases its rate to a destination after each successful transmission. Specifically, after successfully transmitting a packet to a particular destination, the source increases its transmission rate to that destination by 0.1%.

3.2 Changes to Current System

To implement this algorithm, changes must be made to current source nodes and switches. The source nodes have two significant changes. First, the source node must be able to accommodate the new congestion packet and be able to back off accordingly when one is received. Second, the source node needs to be modified to buffer packets at the data link layer and respond to ACKs as they arrive.

The network Ethernet switches also need to be changed. First, the switches require additional memory on each outgoing port to allow for the send and ACK buffers. The delay-bandwidth product, which is the product of link rate and round trip time to the next switch, will equal the maximum amount of data required to be buffered in a switch; buffers will be small since the algorithm uses the hop-by-hop round trip time rather than the longer end to end round trip time.

Next, switches will need to be modified in order to create the ACK packets. Finally, switches will need to be modified to create congestion notification packets when the threshold limit is reached in the buffer of a specific output port. The switch will also need to be modified to be able to send congestion notification packets back to a source node.

3.3 Advantages of the Algorithm

The proposed solution is better than then the current system in a few major ways. First, the algorithm uses a Hop-by-Hop Back Pressure algorithm which reduces the maximum buffer sizes needed at any node or switch, as mentioned above. Another reason to use such an algorithm is because it allows the source nodes to be informed of the congestion much earlier than in TCP. This particular part is close to what InfiniBand implements, but differs since in the InfiniBand solution the entire network must be known to figure out congestion. With the solution proposed, switches can work just as they always have and they do not have to have a controller that knows of the entire system.

Secondly, InfiniBand calculates congestion based on each flow and the path that it will take. The sum of the flow through the system must be less than the amount of credits allowed for that flow. In the InfiniBand system, each sending source node must get credits to be able to inject packets into the network. With the proposed solution, the Back Pressure algorithm is used. This allows for no virtual flows. Instead of keeping track of all flows, like in InfiniBand, congestion is kept track by allowing a buffer to fill up and stop receiving packets. Once a congested link stops receiving packets the other links that send to that specific link will begin to get filled up and congested as well. This way no packets are lost in the system and nothing needs to be added to keep track of the system.

Next, the RED algorithm is used to avoid punishing sending nodes to all destinations when congestion occurs. In a TCP based system, when congestion occurs a sender is punished to all destinations; in the proposed implementation, a random source node that is sending to that congested link is punished only to that destination. This way if the sending source node sends to multiple destinations, then it can send normally to the rest of the destination nodes and only slow down to that given destination.

4 Simulation

To analyze the performance of our algorithm, we used software to simulate a network. These simulations were created using CSim. CSim allows for simulating different network configurations. It has feature for keeping track of buffers and average buffer sizes. It also keeps track of utilization of the links and has prebuilt reports that are generated automatically to allow the user to see the results of the simulation.

4.1 Implementation

The first simulation included thirty source nodes sending through two switches to a single destination node as shown in Figure 4. This simulation allows us to study the stability of our algorithm and its effects on the two switches in terms of utilization and buffer sizes.

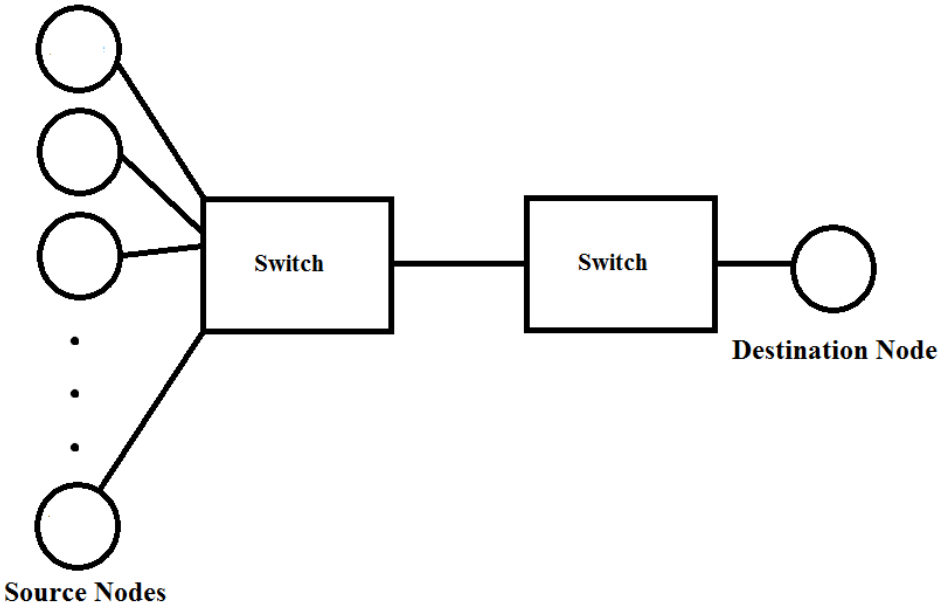

Figure 4: Topology of simulations 1 and 2

Each of the source nodes were given a rate of about 12-20 packets per second to inject traffic into the network and created packets based on that rate. The links in our simulated network were set to be 1 Gb/s with a fixed packet size of 1500 bytes. At this rate a switch could handle about 716K packets per second. Each source node transmits at the same rate with an aggregate rate between 50% and 85% of the total link rate.

Hop-by-hop latency was set to be 200 microseconds as a conservative estimate. According to [20] average latency with 1G Ethernet is greater than 50 microseconds. For the simulation the RED minimum threshold was set to 5 and the RED maximum threshold was set to 15 with a PMAX of 0.1 [20]. If the average is greater than the maximum threshold, the packet is always dropped and a congestion packet is sent to the sender [20].

The second simulation analyzes the effects of congestion on the algorithm. This simulation also uses the network topology shown in Figure 4. To cause congestion one of the source nodes is configured to send at 100% of link speed for an entire second. This caused the sum of the 30 nodes to exceed the available channel capacity on the 1 Gb/s link between the first and second switch.

While the second simulation looked at the congestion of the first switch, the final simulation looks at the effect of congestion further downstream from the source nodes. This final simulation has thirty source nodes send through four switches to a single destination node. Aside from adding the additional two switches, this simulation also added two source nodes attached to the final switch. This topology can be seen in Figure 5. In order to cause congestion this final simulation has the link rate of these links going into the final switch equally at 50% of the link rate for a total of 150% of the outgoing link rate.

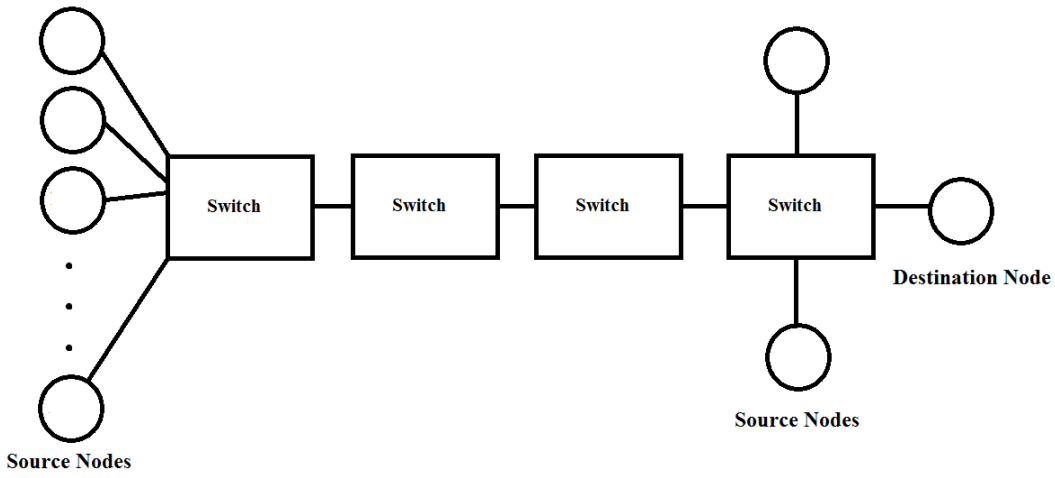

Figure 5: Topology of simulation 3

In addition to using different network topologies described earlier, we ran these simulations with various latencies between hops in order to see the effects of latency on utilization in the network. The hop-by-hop latencies ranged from 200 microseconds to 200 milliseconds.

4.2 Results

Based on our simulations, our results show our algorithm is reliable and stable. We have tracked a number of variables such as end-to-end latency, sizes of the buffers on every source node and switch, and link utilization.

We ran three different simulations using two different network topologies. The results shown in figures 6, 7, 8, and 9 are based on the same network topology (see Figure 4). Figures 6 and 7 present the end-to-end latency of packets going across the network without congestion (Figure 6) and with congestion

(Figure 7). This latency includes the buffering of packets at the source in addition to the network latency. Figure 9, while also is based on a network with congestion, shows the latency once a packet enters the network. These figures show that the system remains stable even when impacted by congestion.

Figure 7 exhibits the average end-to-end latency of all 30 sources. Since one source is heavily congested, the average latency is almost two orders of magnitude greater than shown in figures 6 and 9. Figure 8 shows the same results as Figure 7 except the average does not include the node that is sending at 100% of the link rate and causes congestion. Figure 8 shows that even in a congested network, the non-congested nodes are stable and fairly share the network.

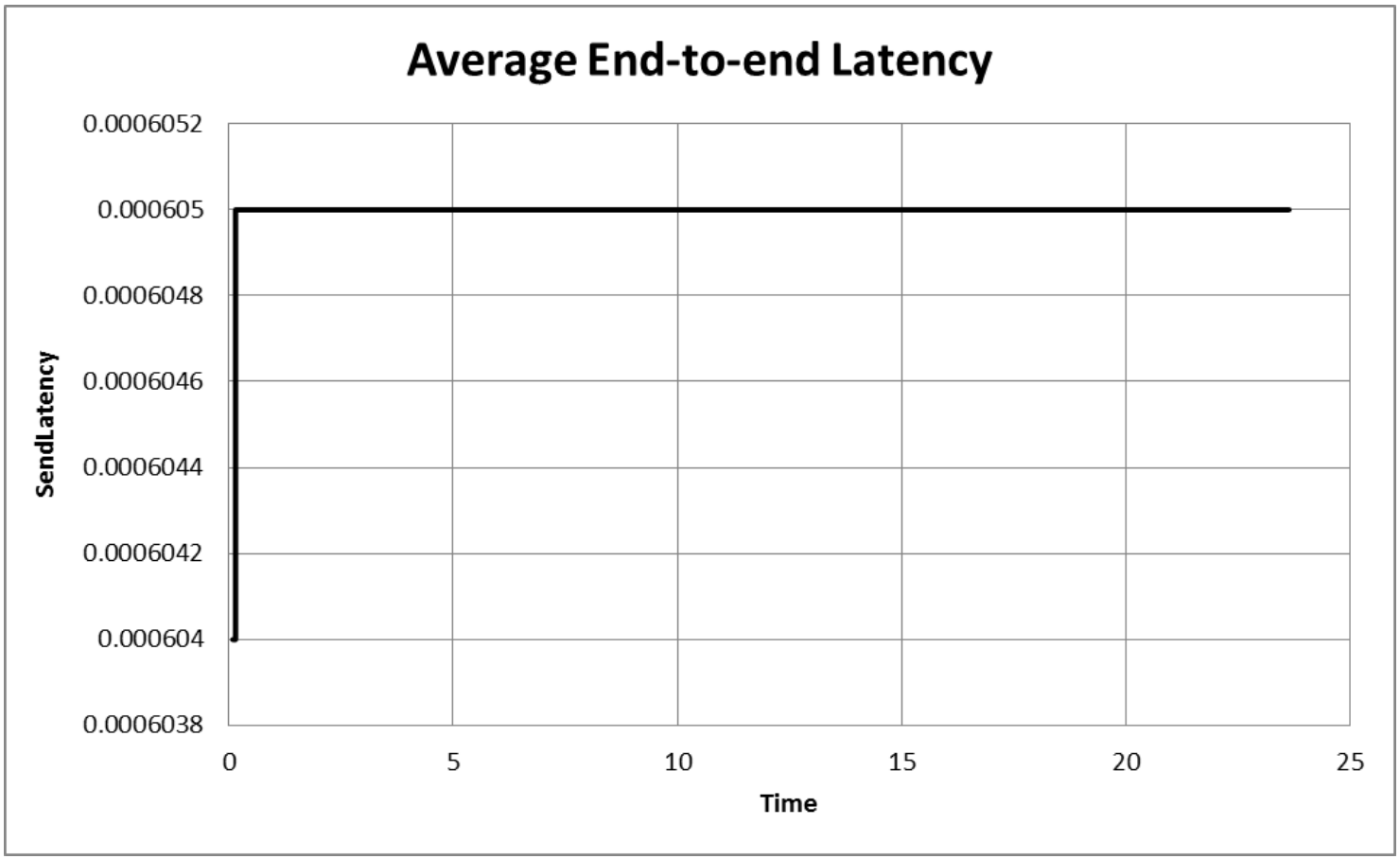

Figure 6: Latency of packets from end to end in a system with no congestion (topology Figure 4)

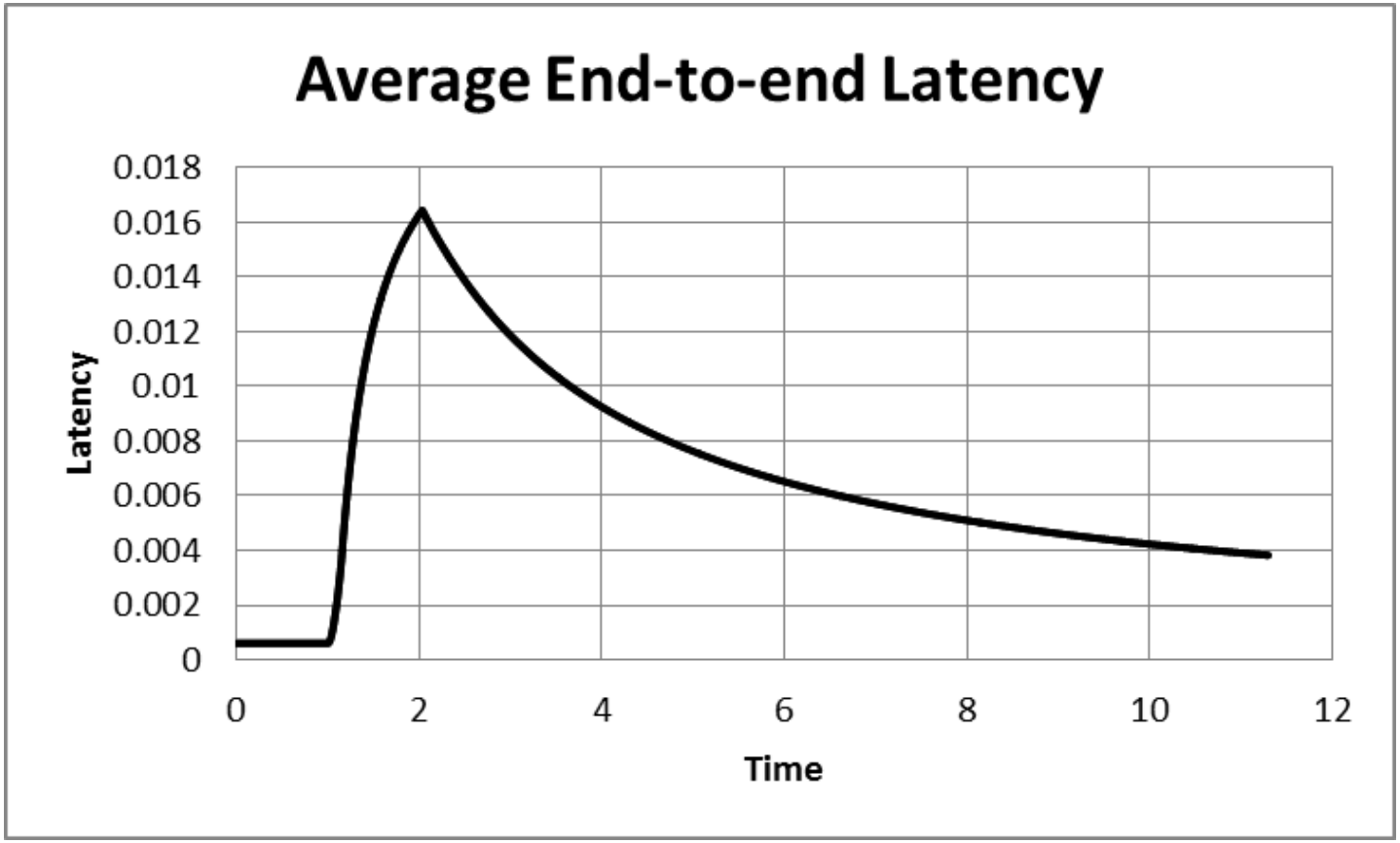

Figure 7: Latency of packets from end to end from generation in a system with congestion (topology Figure 4)

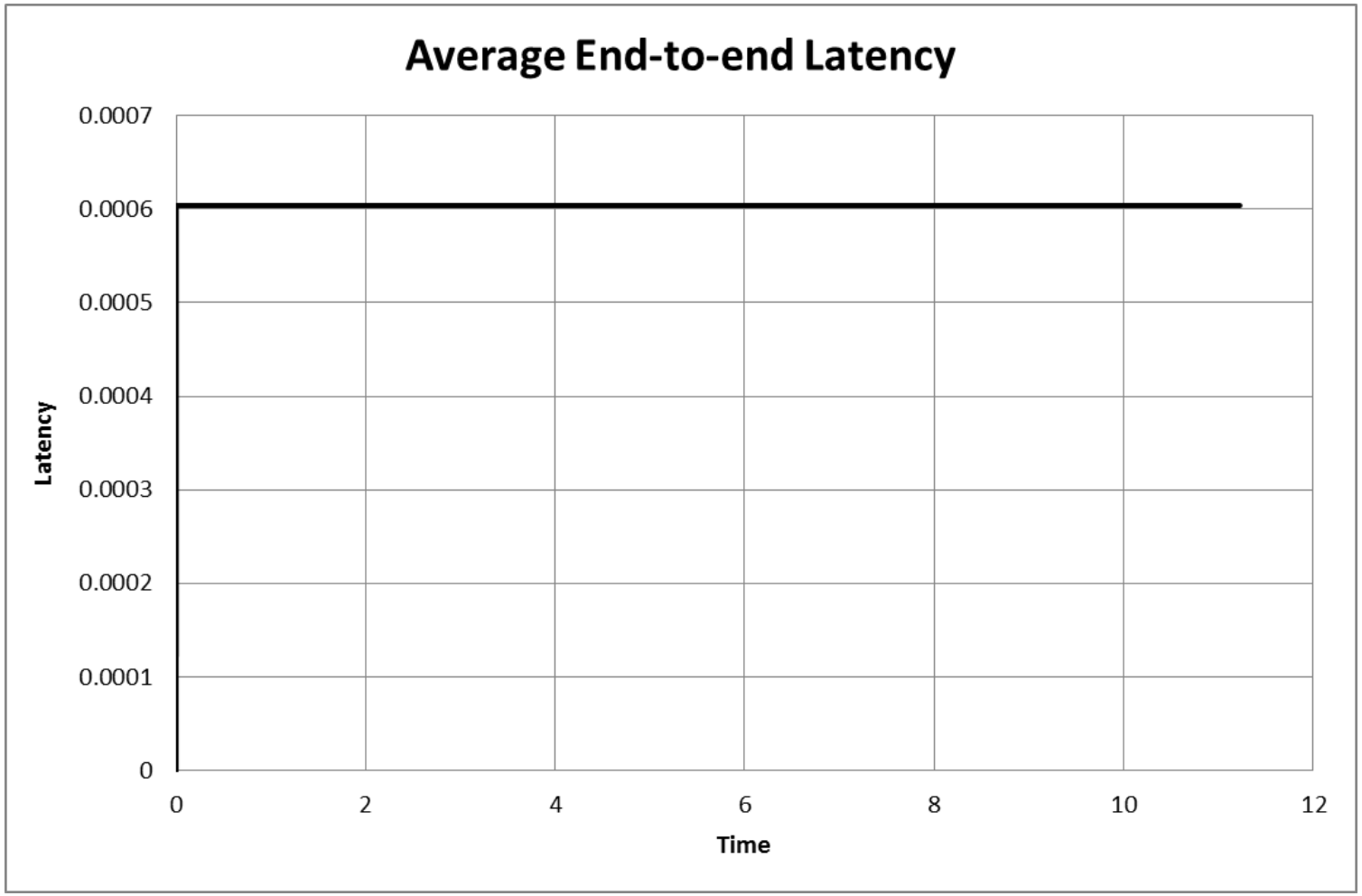

Figure 8: Latency of packets from end to end from generation in a system with congestion (topology Figure 4) without the congested source

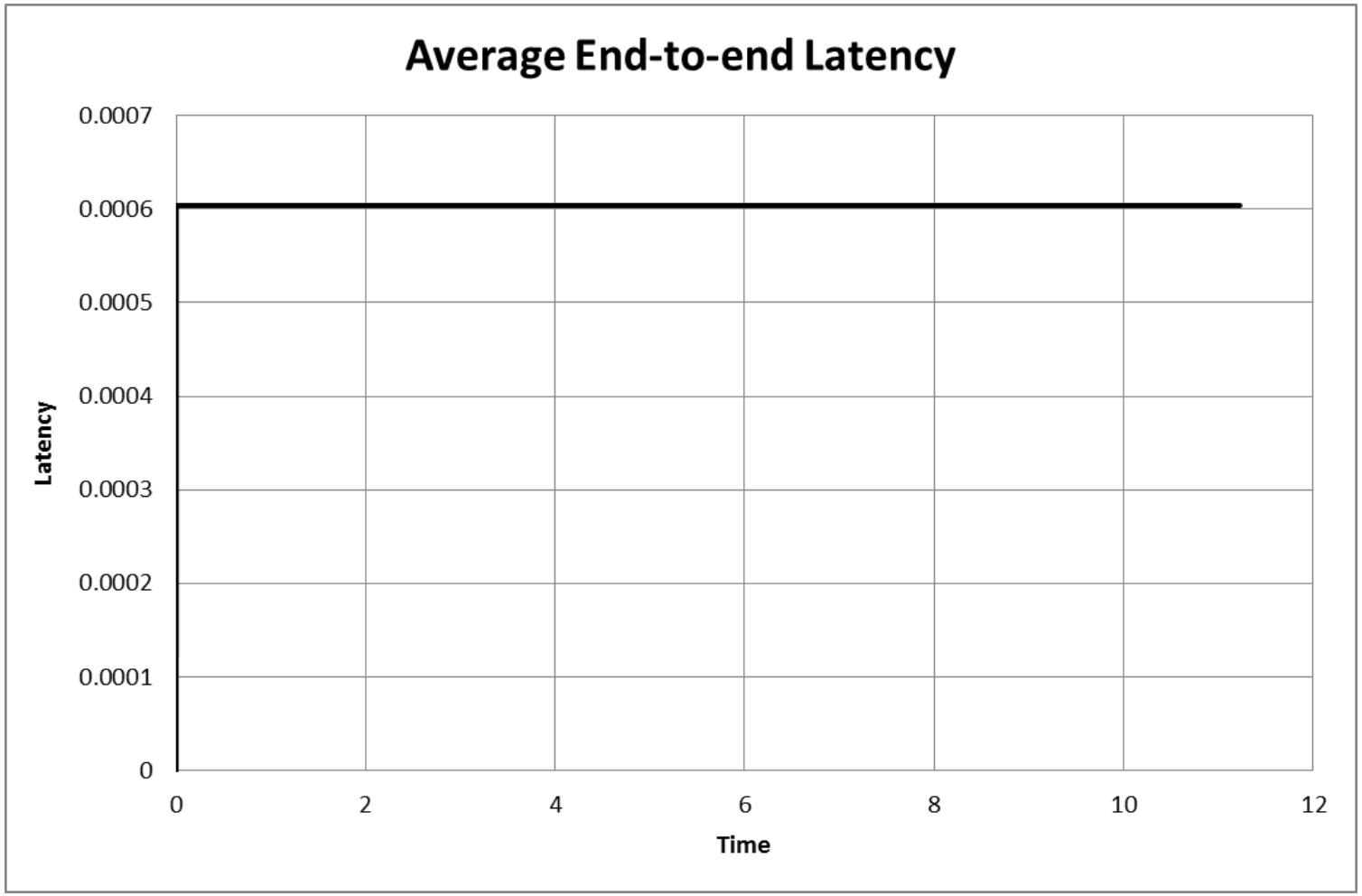

Figure 9: Latency of packets from end to end in a system with congestion after being injected into the network (topology Figure 4)

Figure 10 (simulation without congestion) and Figure 11 (simulation with congestion) show the ACK buffer sizes over time. These figures show that the ACK buffer on each of the source nodes stay constant in networks with or without congestion. Both simulations yielded small ACK buffer sizes on each source node and these buffers did not increase over time. As expected, the second simulation shows that the node that injects heavy traffic into the system has its buffer increase, but decreases once the node resumes transmission at a lower rate.

Figure 10 shows all thirty source nodes converging to a single ACK buffer size and staying there throughout the simulation in a system with no congestion. Based on the round-trip time between the source node and the switch, the ACK buffers should be in the range of 8-10 packets which is consistent with Figure 10.

Figure 11 shows the node that sent at a 100% of line speed had a drastically larger ACK buffer size. Later, once its packet sending rate decreased, the node's ACK buffer decreased in size over time. Figure 11 also shows that the remaining sender node's ACK buffers remain within the expected range of 8-10 packets.

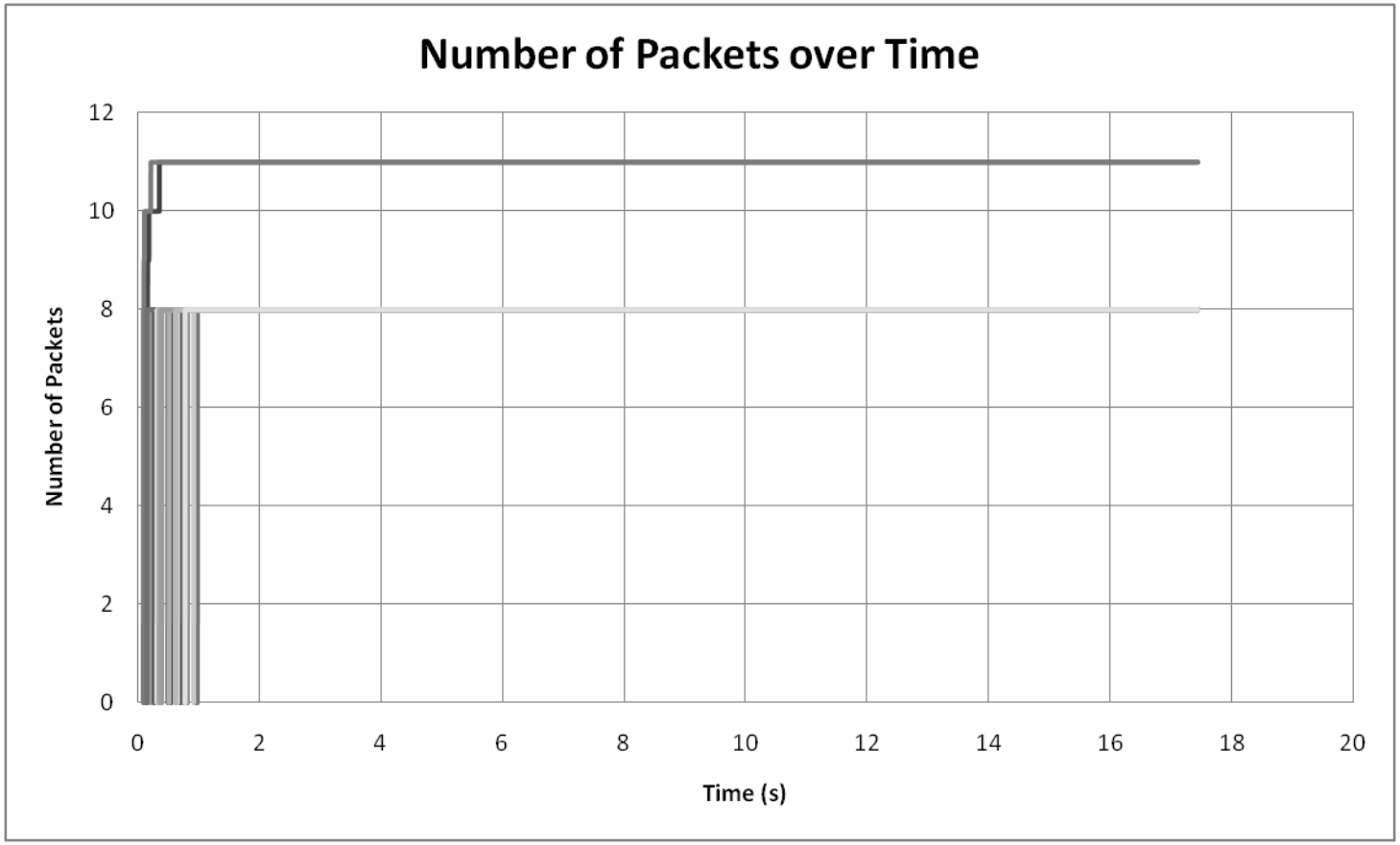

Figure 10: Number of packets in the ACK buffer of each source node in a system with no congestion (topology Figure 4)

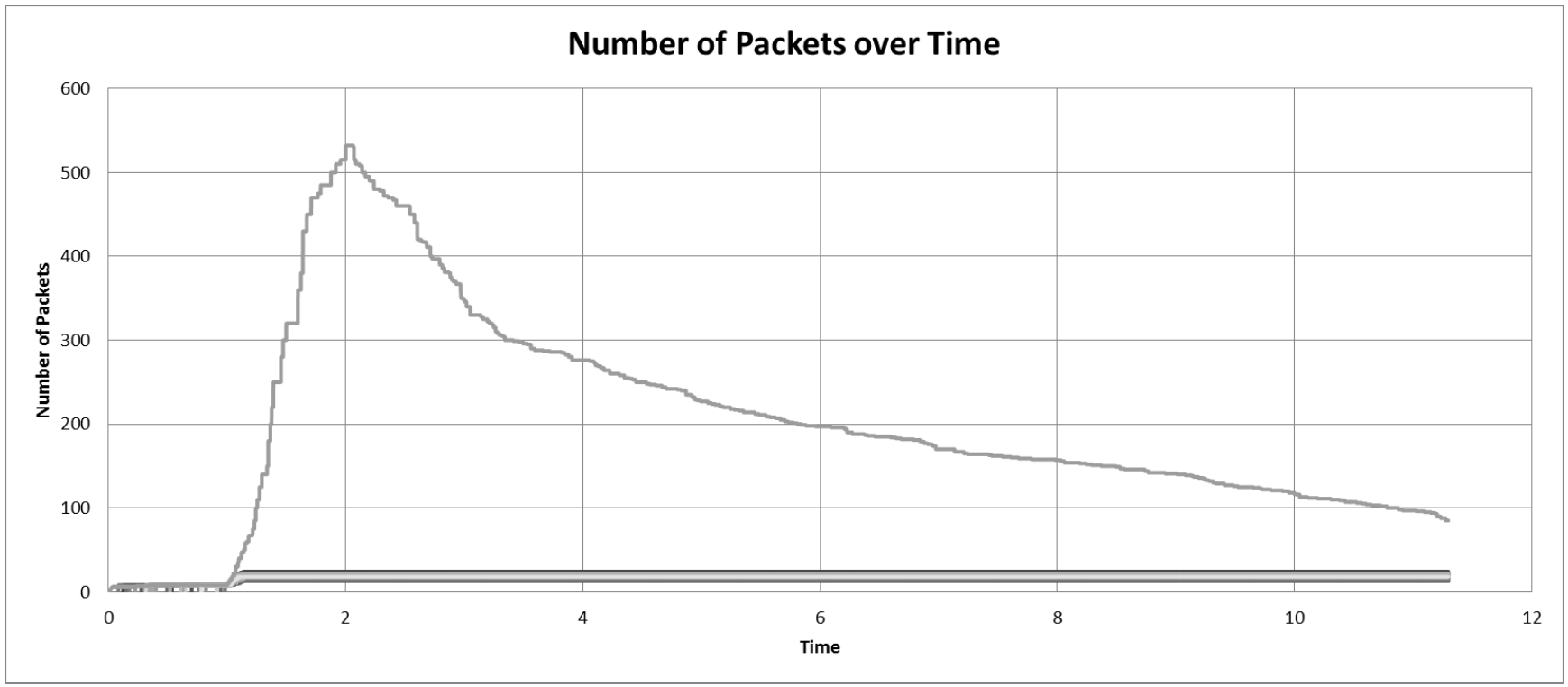

Figure 11: Number of packets in the ACK buffer of each source node in a system with congestion (topology Figure 4)

Figure 12 looks at the ACK buffer on each switch in a system with no congestion. This figure shows that without congestion that the ACK buffer size stays constant. The ACK buffer size on switch 1 is higher, in the range of 12-14 packets, than switch 2. This is due to the fact that our RED algorithm is tuned to keep an average transmission rate of 55% of total link rate. Since the source nodes are attempting to send at 80% if link rate, the RED algorithm on switch 2 is discarding packets. These discarded packets are therefore not removed from switch 1's ACK buffer until they are retransmitted and these retransmissions acknowledged by switch 2.

While not shown, in a network with congestion, the average switch ACK buffer graph is identical to Figure 12. Combining figures 11 and 12 show that while a node that heavily utilizes the network will have large source buffers, the network remains stable and performs similarly to an uncongested network.

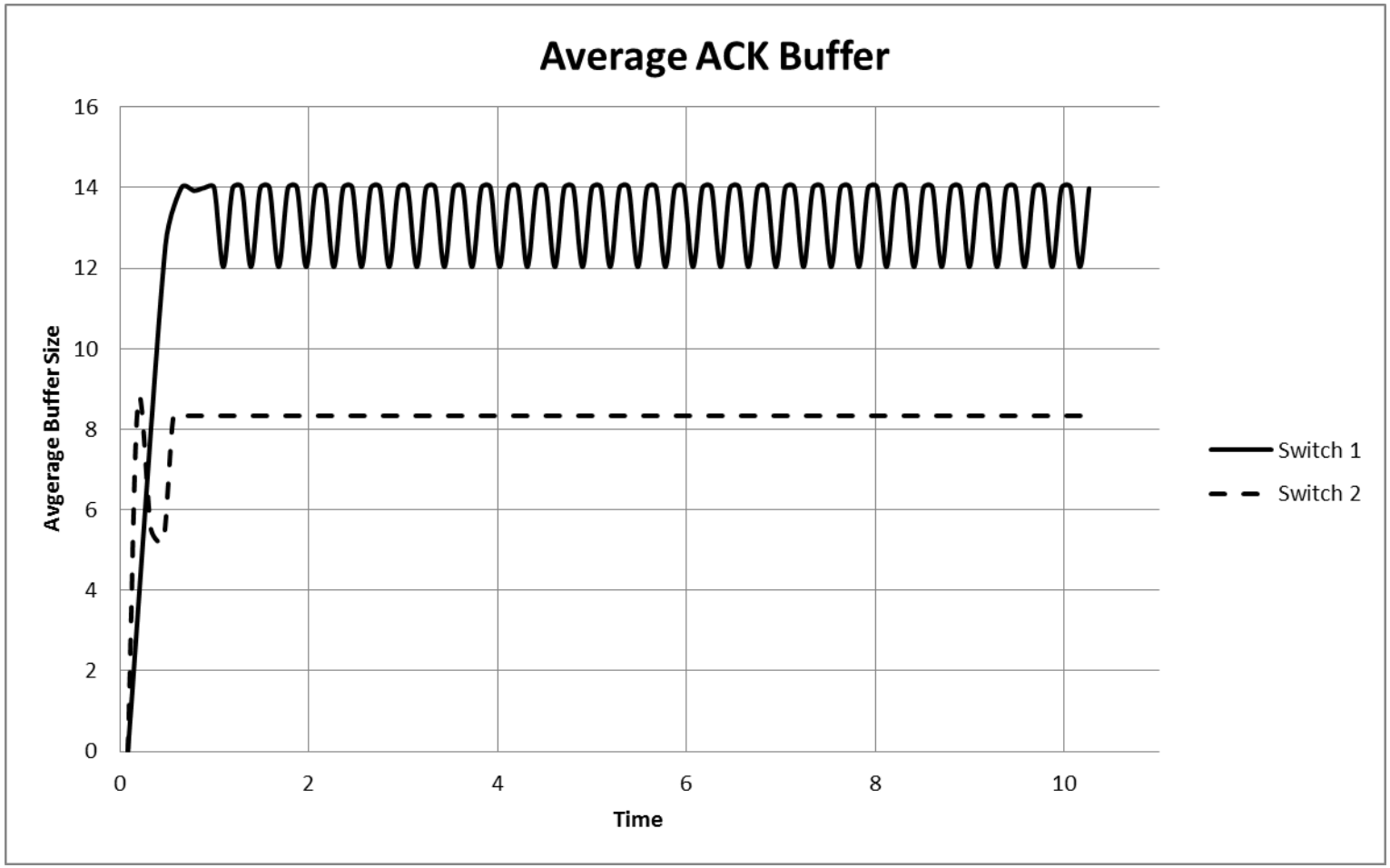

Figure 12: Average buffer size in the ACK buffer over time in a system with and without congestion (topology Figure 4)

Figure 13 (without congestion) and Figure 14 (with congestion) present the size of the send buffer on the two switches. These figures show that with or without congestion the send buffers remain small and vary between 0 and 3 packets.

Figure 15 (no congestion) and Figure 16 (congestion) present the average utilization on the link between switch 1 and switch 2. These graphs show that the utilization approaches 80% for both congested and non-congested simulations.

Combining figures 15 and 16 with the earlier figures which analyze the buffer sizes on the switches shows that in congested and uncongested conditions the switch's buffer sizes remain small and the utilization are consistent and approach 80%.

Figure 17 shows the sending rate of the congested node in the first topology. This graph show promising results, since AIMD is performing correctly and keeping the node sending at a rate at which the network stays uncongested and allows the node to send at the maximum rate to reduce its buffer.

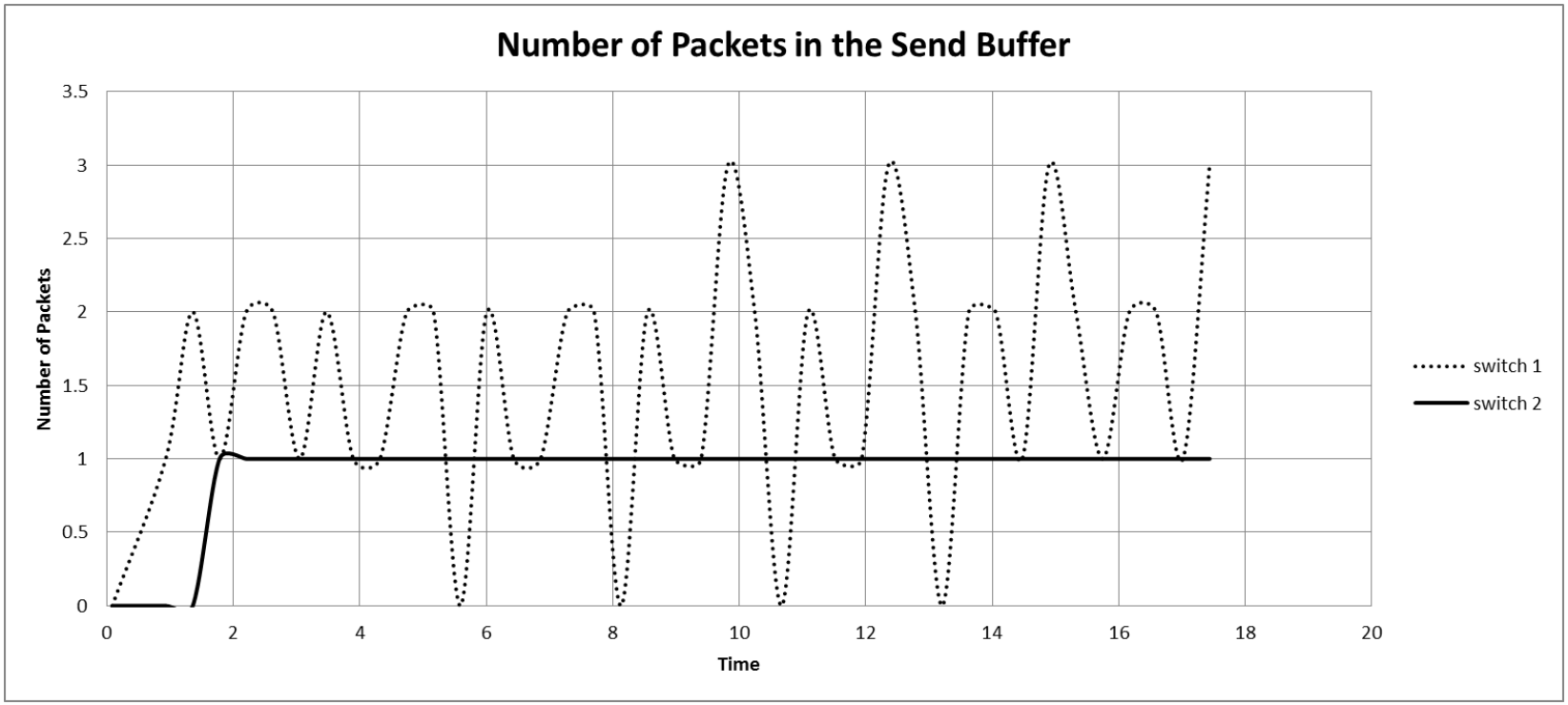

Figure 13: Number of packets in the send buffer at a given time in the switches in a system with no congestion (topology Figure 4)

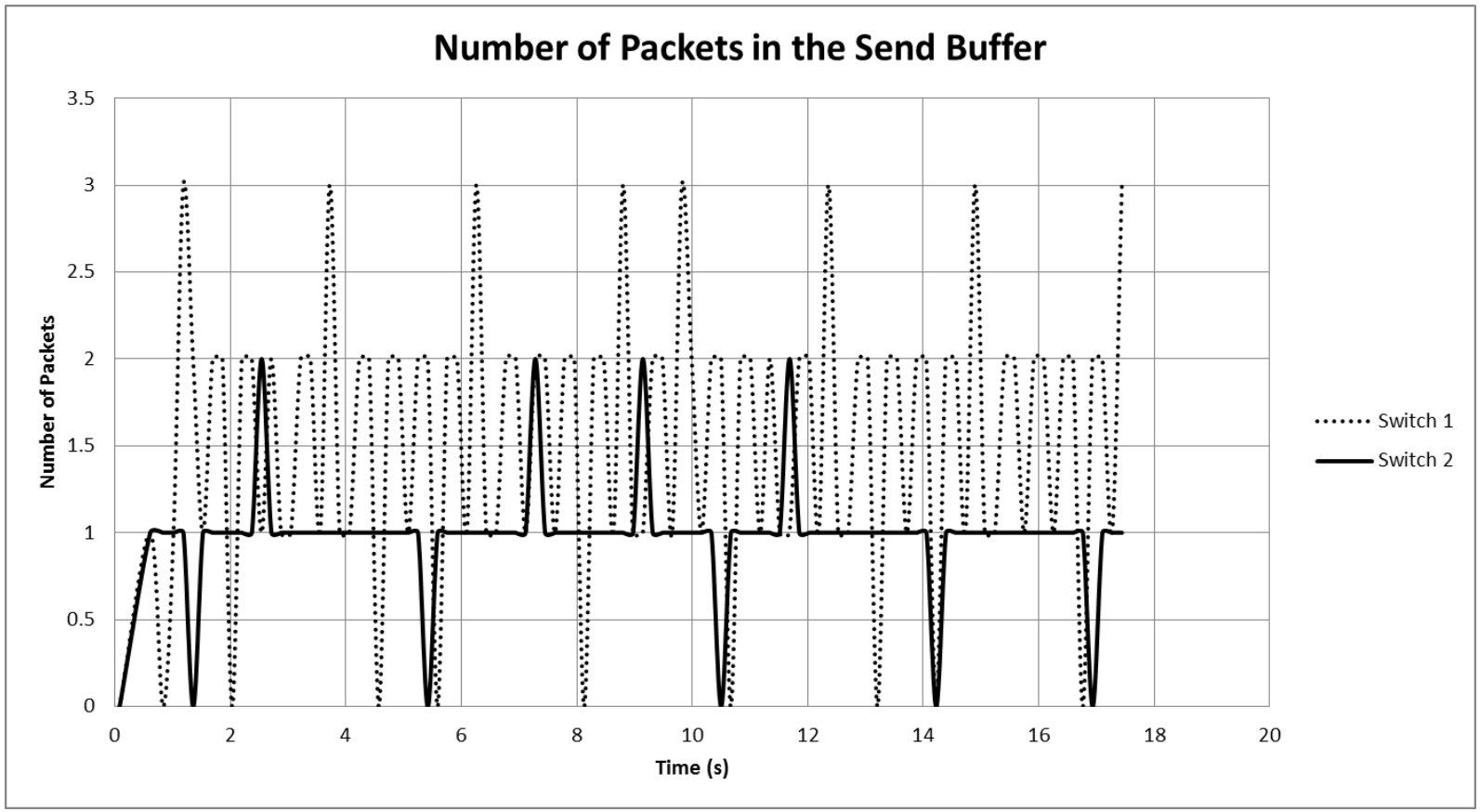

Figure 14: Number of packets in the send buffer at a given time in the switches in a system with congestion (topology Figure 4)

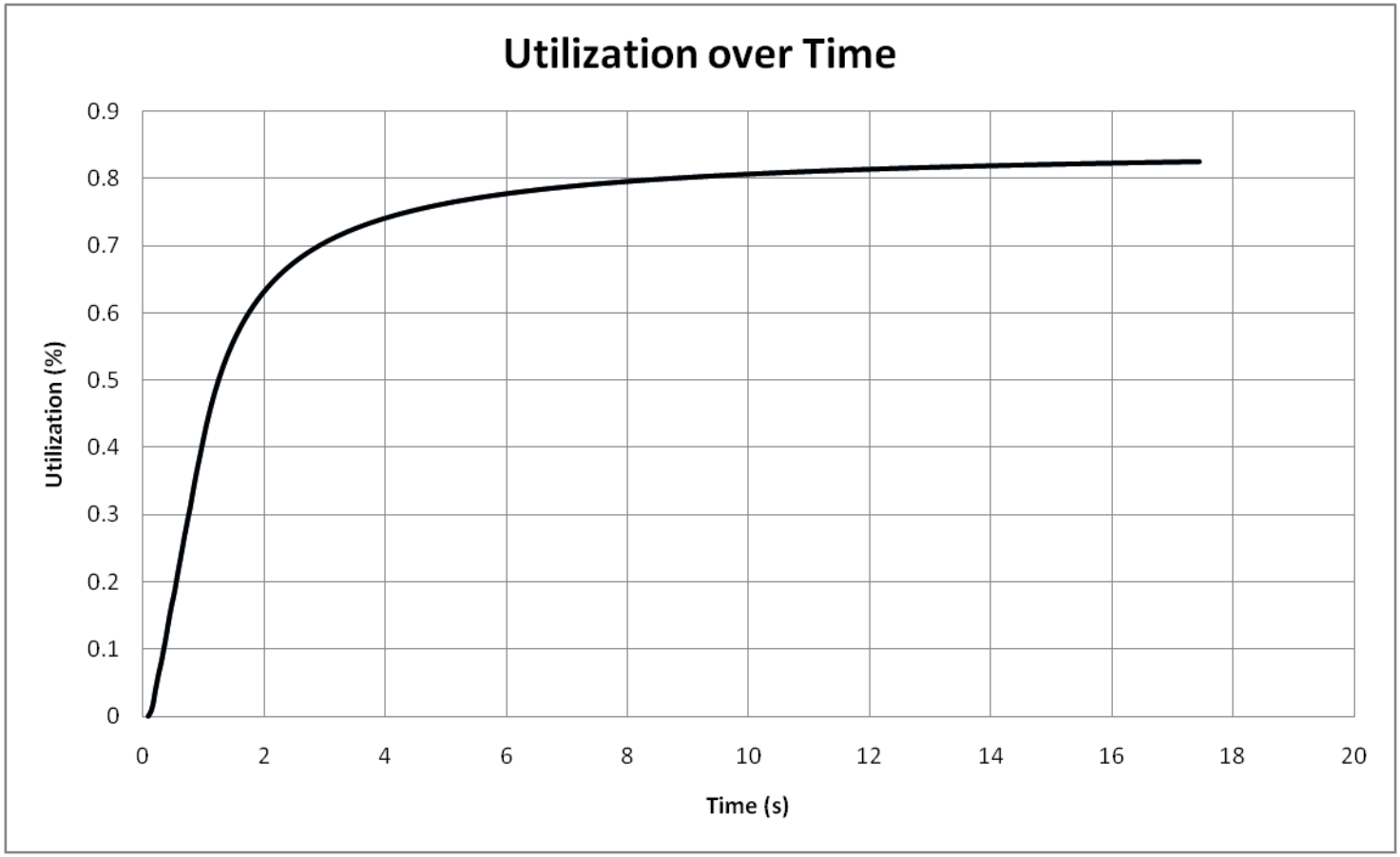

Figure 15: Utilization between switch 1 and switch 2 in a system with no congestion (topology Figure 4)

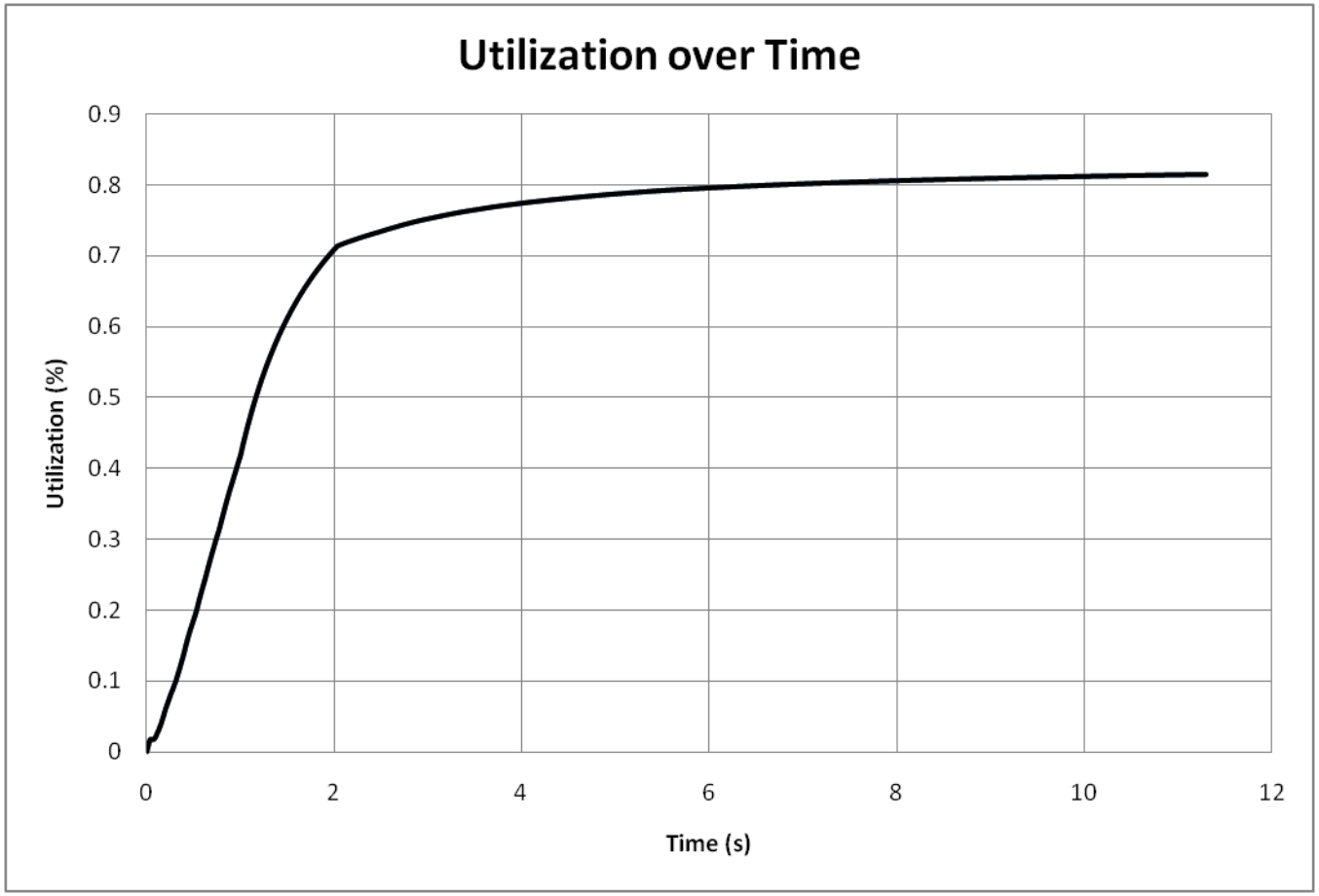

Figure 16: Utilization between switch 1 and switch 2 in a system with congestion (topology Figure 4)

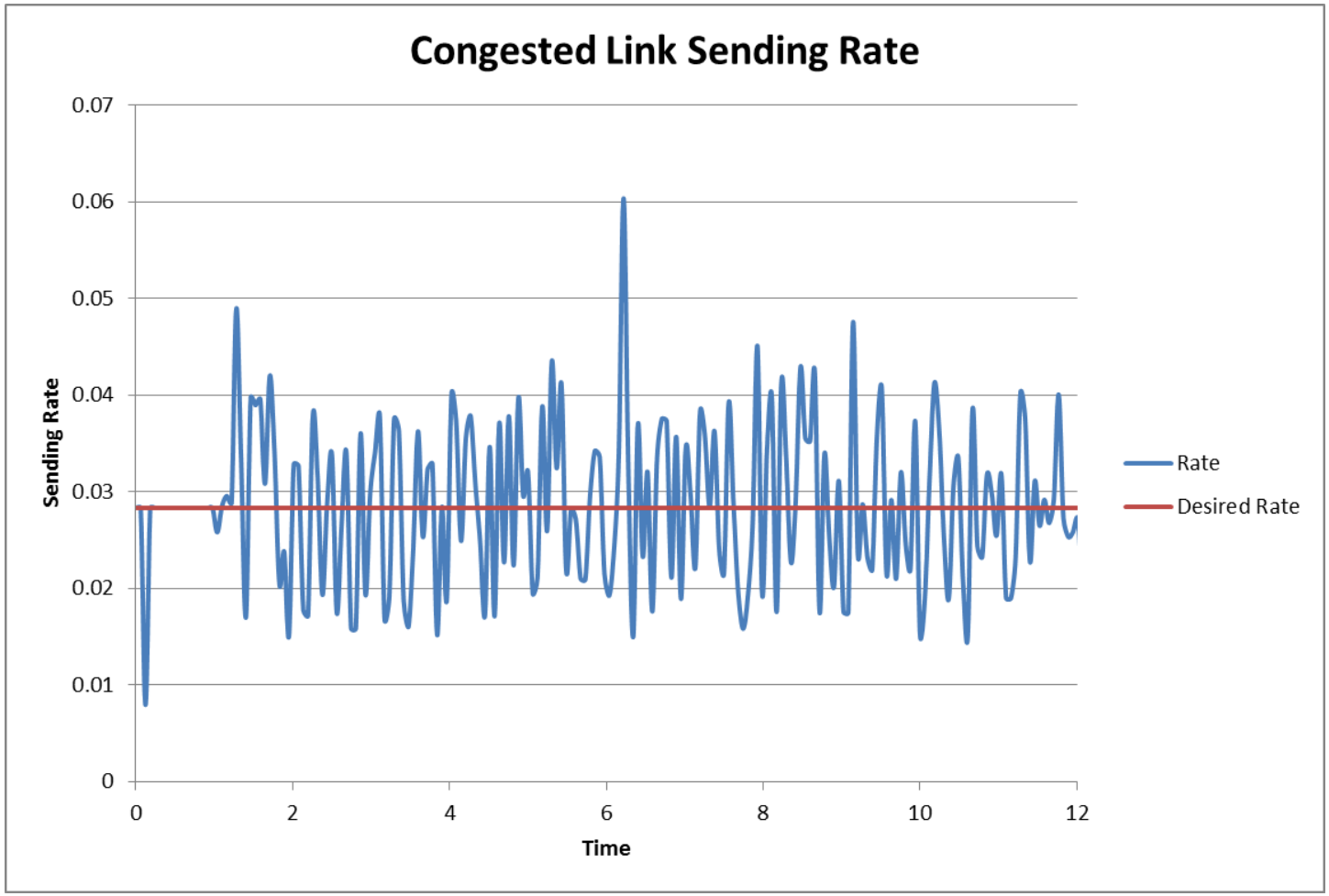

Figure 17: Sending rate of the congested node (topology Figure 4)

Figure 18 and Figure 19 show the ACK and send buffer on the final downstream switch in topology Figure 5. These graphs show that even with many switches in a congested system, the algorithm performs well. The switch's buffers perform exactly the same as in topology Figure 4.

Some initial tests were also performed changing the RED threshold values, the latency of the network, and the packet sizes. As shown in Figure 20, our tests showed that the algorithm is sensitive to latency. As latency between nodes increased the utilization on the links decreases and the ACK buffer size increases. As Figure 20 shows, for hop-by-hop latency less than 20 milliseconds, link utilization remains above 80%. Since our approach only depends on the latency on a single link and not the end-to-end we can expect our algorithm to perform well in a real-world network.

In the final simulation, the congestion and latency had major effects on the upstream switch. As shown in Figure 20, hop by hop latency should be kept fairly small in order to allow for maximum utilization. As hop by hop latency increases the utilization of the switches decreases. This also shows that to have switches perform optimally, even cases of congestion do not need unrealistic latencies. The utilization only becomes a major problem when hop-by-hop latency is about 700 milliseconds.

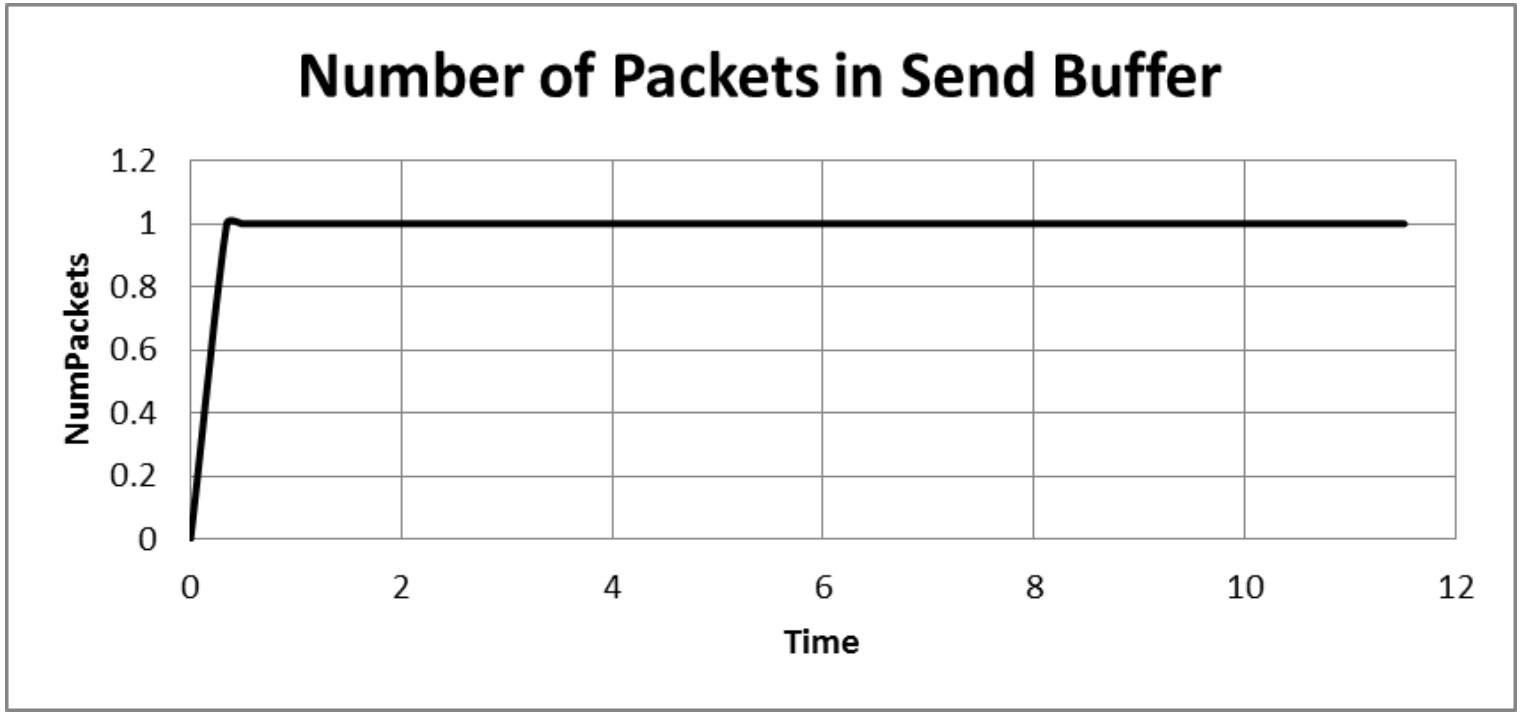

Figure 18: Number of packets in the send buffer at a given time in the switches in a system with congestion (topology Figure 5)

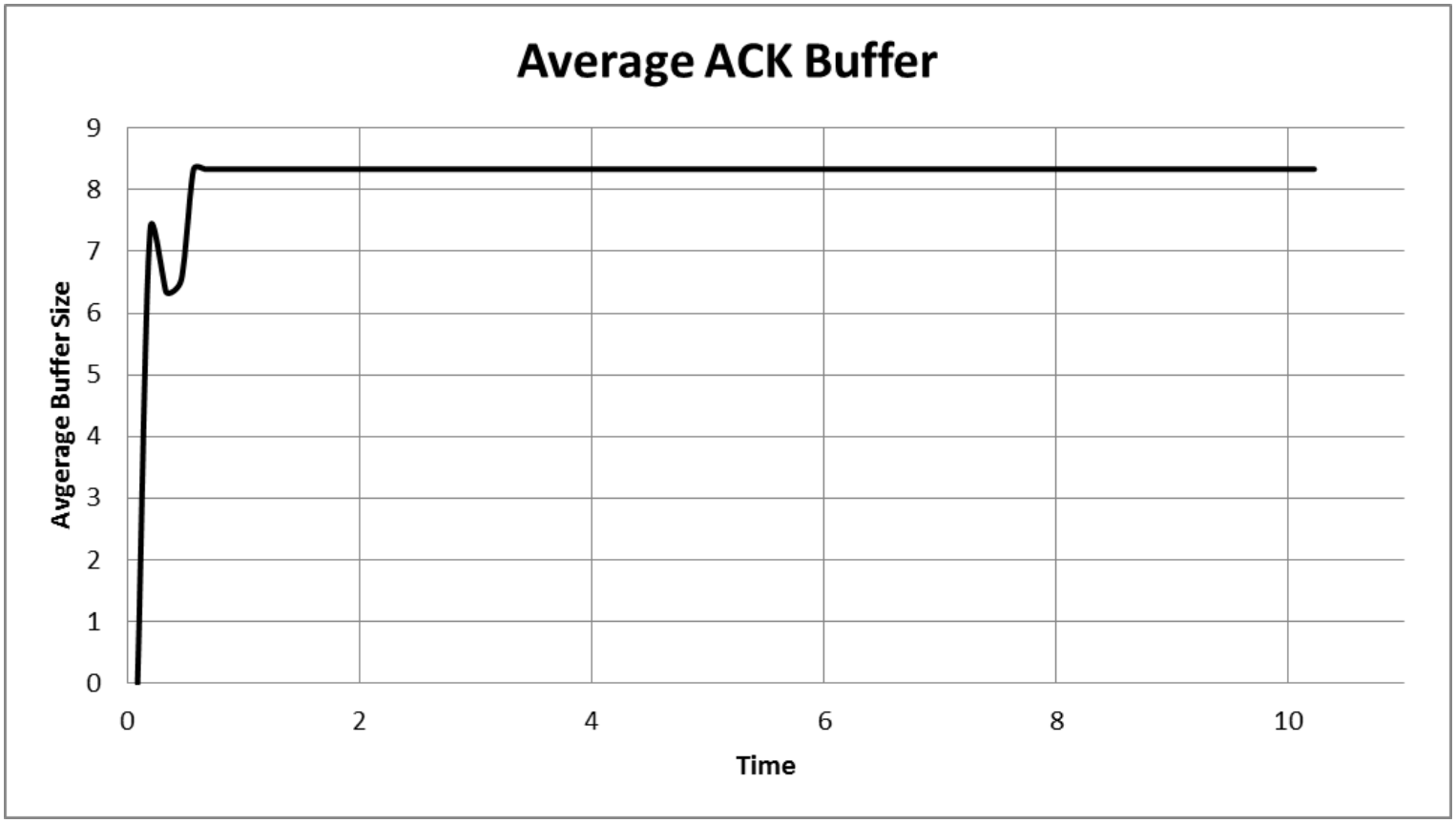

Figure 19: Average buffer size in the ACK buffer over time in a system with and without congestion (topology Figure 5)

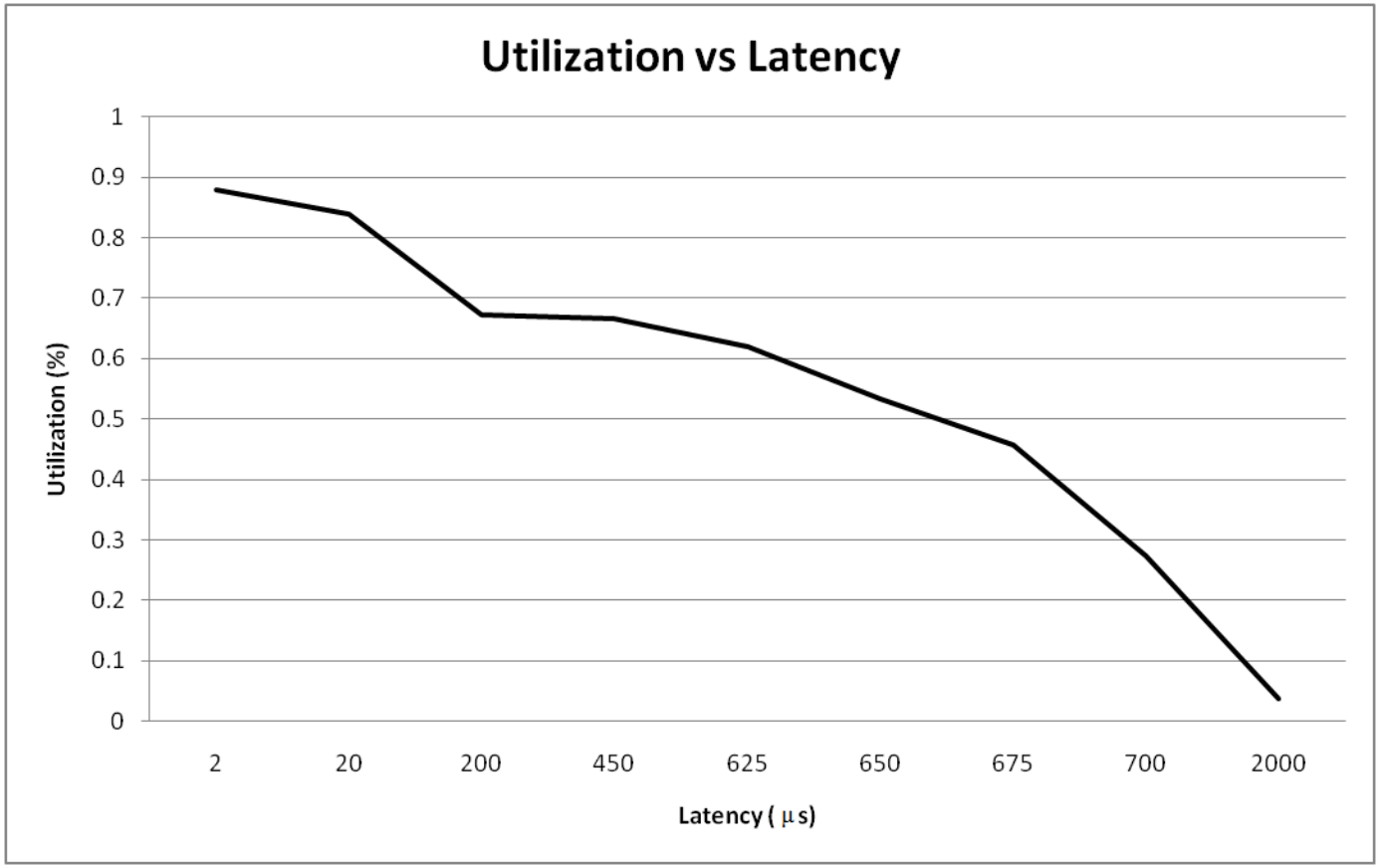

Figure 20: Utilization over hop by hop latency (topology Figure 5)

5 Future Work

There is still work to be done on this topic to determine how this new implementation will work in the real world. The current implementation of the algorithm slows down the sending rate based on the source and destination MAC addresses. Other ways to decrease the rate of the sender could be examined to increase the efficiency of the implementation.

Another topic that would need to be explored is the implication of the algorithm when a switch stops working. Currently with TCP, there are always two copies of a packet within the system so that if a switch does go down and loses all packets currently on the switch, then the sender can resend the packet to the destination. In our approach, only one switch or node has the packet at any given point. If a switch were to break or shut down, then all packets within the buffer of that switch would be lost. In this case the sender would never know of this problem since it has received an ACK packet from its next hop and assumes that the packet will reach the given destination.

This system does not provide for any traffic differentiations. This would allow different packets to be sent with different priority levels and have the algorithm back off on lower priority source nodes rather than any random source node.

Finally, the system would need to be implemented into an actual network to figure out the correct threshold values, the correct alpha value, and to create real life tests to ensure the implementation would work. Implementing the solution onto actual switches would also allow a side-by-side comparison of latency of the proposed algorithm and TCP systems both with and without congestion.

6 Conclusion

Currently TCP controls congestion within Ethernet based networks. In low latency networks, the faster congestion is identified and controlled the better it is for the system. The biggest problem of TCP in these networks is that its flow control is based on end-to-end round trip time. This causes a delay in congestion identification and response which causes many packets to be dropped and creates a need for them to be resent.

With this proposed algorithm, congestion is identified based on hop-by-hop latency. This allows for congestion to be identified quicker and allows for faster response from the source nodes.

Initial tests of the algorithm in simulation prove that it can be a solution to the problem of unreliable Ethernet and that is allows the system to be faster than the current implementation. This is done mainly by removing the overhead TCP and IP from the route which increases packets going through devices. We are currently limited to the speeds of TCP and IP, and so by removing them, we remove the bottleneck. To do this we would need a new congestion algorithm that can substitute these two protocols and perform well in a system. The algorithm in this paper does this and shows that it can handle networks with and without congestion.

As shown by the implementation, correctly choosing threshold and alpha values for the RED algorithm is important to allow the algorithm to work effectively. The simulation tests gave good results, but would need to be tested in actual networks to determine the viability of the proposed solution in the actual world where unexpected events occur.

We believe the proposed algorithm in this paper will work with a few minor tweaks once implemented on hardware. This can be a solution with faster recovery time and faster average latencies across the network. It has performed in a network with a large burst of traffic while keeping buffers low, latencies of packets end to end constant, and utilization of switches close to optimal.

BIBLIOGRAPHY

[1] Internet Engineering Task Force., "Requirements for Internet Hosts -- Communication Layers." October 1989.

[2] Floyd, Sally and Jacobson, Van., "Random Early Detection Gateways for Congestion Avoidance." IEEE/ACM Transactions on Networking, August 1993.

[3] Leung, I.K and Muppala, J.K., "Packet marking strategies for Explicit Congestion Notification (ECN)." *Performance, Computing, and Communications, 2001. IEEE International Conference.* April 2001, pp. 17-23. URL: http://www.ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=918631&isnumber=19865.

[4] Mathis, M, et al., *TCP Selective Acknowledgment Options.* [RFC 2018] s.l. : Network Working Group, October 1996.

[5] Srijith, K.N., Jacob, L. and Ananda, A.L., "Worst-case performance limitation of TCP SACK and a feasible solution." s.l. : IEEE, 2002. Communication Systems, 2002. ICCS 2002. The 8th International Conference. Vol. 2, pp. 1152-1156. URL: http://www.ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1183313&isnumber=26554.

[6] Pentakalos, Odysseas., An Introduction to the InfiniBand Architecture. *O'Reilly.* [Online] O'Reilly, February 4, 2002. http://www.oreillynet.com/pub/a/network/2002/02/04/windows.html.

[7] Eracar, Y.A., "Benefits of LRU-centric fibre channel testing." *IEEE.* September 17-20, 2007, pp. 741-750. URL: http://www.ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4374293&isnumber=4374190.

[8] Yunqi, Luo, et al., "BO-ARED: A new AQM algorithm with adaptive adjustment of parameters." 2010. Intelligent Control and Automation (WCICA), 2010 8th World Congress on. pp. 1852 -1857. URL: http://www.ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5554486&isnumber=5553742.

[9] Floyd, Sally, Gummadi, Ramakrishna and Shenker, Scott., *Adaptive RED: An Algorithm for increasing the Robustness of RED's Active Queue Management.* [PDF] August 2001. URL: http://icir.org/floyd/papers/adaptiveRed.pdf.

[10] Luo, Yunqi, et al., "BO-ARED: A new AQM algorithm with adaptive adjustment of parameters." 2010. Intelligent Control and Automation (WCICA), 2010 8th World Congress on. pp. 1852 -1857.

[11] Ippoliti, Dennis, Zhou, Xiaobo and Zhang, Liqiang., "Packet Scheduling with Buffer Management for Fair Bandwidth Sharing and Delay Differentiation." 2007. Computer Communications and Networks. pp. 569-574.

[12] Santhi, V and Natarajan, A.M., "A new approach to Active Queue Management for TCP with ECN." 2009. Advanced Computing, 2009. ICAC 2009. First International Conference on. pp. 76-81.

[13] Kim, Beomjoon, Kim, Dongmin and Lee, Jaiyong., "Lost retransmission detection for TCP SACK." Communications Letters, IEEE, September 2004, Issue 9, Vol. 8, pp. 600-602. 1089-7798.

[14] Barroso, Jose M., "UETS/EFR: A Highly Scalable Ethernet Service Delivery Over Optical Transport Infrastructure."

[15] Barroso, Jose M. and Fernandez, Guillermo I., "Ethernet Fabric Routing (UETS/EFR) - A Hierarchical, Scalable and Secure Ultrahigh Speed Switching Architecture." 2006. 25th IEEE International Conference on Computer Communications. Proceedings.

[16] Katabi, Dina, Handley, Mark and Rohrs, Charlie., "Congestion control for high bandwidth-delay product networks." SIGCOMM Comput. Commun. Rev., New York : ACM, August 2002, Issue 4, Vol. 32, pp. 89-102. http://doi.acm.org/10.1145/964725.633035. 0146-4833.

[17] Lee, Jaesung, Lee, Hyuk-Jae and Park, Kyoung., "An Efficient Implementation of the InfiniBand Link Layer." IEEE, November 2003, pp. 355-358. 0-7803-8182-3.

[18] Turner, Yoshio, Santos, Jose Renato and Janakiraman, G. (John)., "An Approach For Congestion Control In Infiniband." May 2002.

[19] Tan, Yu-an, et al., "A High-Throughput Fibre Channel Data Communication Service." 2005. Parallel and Distributed Computing, Applications and Technologies, 2005. PDCAT 2005. Sixth International Conference on. pp. 975-978.

[20] Weber, Dave, et al., "New Approaches to Ultra-Low Latency and Creating Dynamic Insights." [Online] September 20, 2010. http://www.flaggmgmt.com/hpc/Session%205-IBM%209-20-10.pdf.

[21] Lassila, Pasi E. and Virtamo, Jorma T., "Modeling the Dynamics of the RED Algorithm." London : Springer-Verlag, 2000. QofIS '00: Proceedings of the First COST 263 International Workshop on Quality of Future Internet Services. pp. 28-42. 3-540-41076-7.

[22] Hedrick, C., *Routing Information Protocol.* s.l. : Network Working Group, June 1988. http://tools.ietf.org/pdf/rfc1058.pdf.

[23] Marquart, Jane., "Building a Reliable Onboard Network with Ethernet: a GSFC Prototype." [Power Point Presentation]. June 2004.

[24] Mellanox Technologies., *The Case for InfiniBand over Ethernet.* Mellanox Technologies. 2008. Whitepaper.

[25] Uchida, Tomohisa., "Hardware-Based TCP Processor for Gigabit Ethernet." IEEE Transactions on Nuclear Science, June 2008, Issue 3, Vol. 55, pp. 1631-1637.

[26] Kaashoek, M. Frans., *An Efficient Reliable Broadcast Protocol.* Vrije Universiteit Amsterdam, The Netherlands. Masters Thesis.

[27] Schmid, Stefan., "A TCP with Guaranteed Performance in Networks with Dynamic Congestion and Random Wireless Losses." ACM, August 2006.

[28] Ishihara, Tomohiro., "Carrier-Grade Ethernet Switch for Reliable Wide-Area Ethernet Service." FUJITSU Scientific and Technical Journal, December 2003, Vol. 39, pp. 234-243.

[29] Falk, Aaron., "Transport Protocol for High Performance." ACM, November 2003, Issue 11, Vol. 46, pp. 43-49.

[30] Dreier, Roland., "InfiniBand and Linux." Linux Journal, March 2005, Issue 8131. http://www.linuxjournal.com/article/8131.

[31] Majumder, Supratik., "Comparing Ethernet and Myrinet for MPI Communication." [Rice University Publication].

[32] Shu, Jiwu., "Design and Implementation of an SAN System Based on the Fiber Channel Protocol." IEEE Transactions on Computers, April 2005, Issue 4, Vol. 54, pp. 439-448.

[33] Ocheltree, Kenneth B., "A Comparison of Fibre Channel and 802 MAC Services." IEEE, 1993, pp. 238-246.

[34] Gigabit Ethernet Alliance., "Gigabit Ethernet 1000 Base-T." [Whitepaper]. 1997.

[35] Kleiman, Steve., "DAFS: A New High-Performance Networked File System."

[36] Mellanox Technologies., "The Case for InfiniBand over Ethernet - The Evolutionary Step for IPC Consolidation over 10 Gigabit Ethernet." [Whitepaper]. April 2008.

[37] Reinemo, Sven-Arne, et al., "An Overview of QoS Capabilities in InfiniBand, Advanced Switching, Interconnect, and Ethernet." IEEE Communications Magazine, July 2006, Issue 7, Vol. 44, pp. 32-38.

[38] Santos, Jose Renato, Turner, Yoshio and Janakiraman, G. (John)., "Evaluation of Congestion Detection Mechanisms for InfiniBand Switches." August 2002.

[39] Meggyesi, Zoltán., Fibre Channel Overview. [Online] Research Institute for Particle and Nuclear Physics, August 15, 1994. http://www94.web.cern.ch/HSI/fcs/spec/overview.htm.

[40] Zheng, Bing and Atiquzzaman, Mohammed., "A framework to determine bounds of maximum loss rate parameter of RED queue for next generation routers." *Network and Computer Applications.* November 2008, Vol. 31, pp. 429-445.